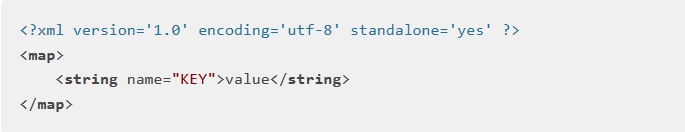
1. **Storage** 
   1. **Shared Preferences**
      1. **Cấu trúc bên trong của Shared Preferences**

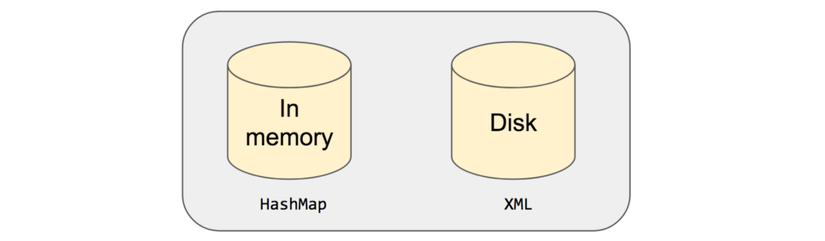
Để có cái nhìn dễ hiểu hơn về Shared Preferences, chúng ta có thể hình dung cấu trúc bên trong **Shared Preferences** là một bộ nhớ **in-memory (RAM)** đặt trên một bộ nhớ **disk-storage (External Storage)**. Mọi thao tác đều đi qua bộ nhớ **in-memory** đầu tiên sau đó mới đến **disk-storage** trong trường hợp cần thiết.

- Bộ nhớ **In-Memory** về bản chất là một **Hash-Map**, điều đó cho phép các thao tác đều có độ phức tạp là O(1) khi chạy.



* **Disk Storage** thực chất là một file xml được cấu trúc như sau:





* + 1. **Sử dụng Shared Preferences**

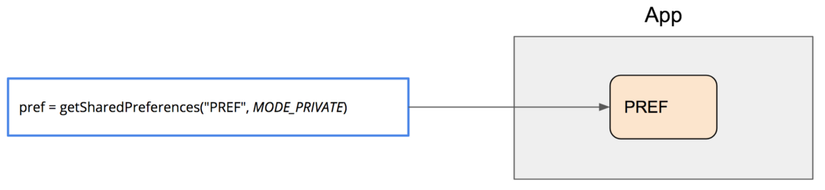
1. **Lấy instance của Shared Pref**

Sử dụng hàm **getSharedPreferences(Name, Mode)** để lấy 1 **instance** của **Shared Preferences** với Name là tên của **Shared Preferences**

* **SharedPreference mode:**
* **MODE\_PRIVATE**: chỉ ứng dụng có thể truy nhập vào.
* **MODE\_WORLD\_READABLE**: các ứng dụng khác có thể đọc được nội dung nhưng không thể sửa hoặc thêm.
* **MODE\_WORLD\_WRITABLE**: Ứng dụng khác có thể đọc/ghi nội dung.
* **MODE\_MULTI\_PROCESS**: Nhiều process có thể cùng chỉnh sửa SharedPreference.

Đoạn code dưới đây sẽ lấy một **instance** của **shared preferences** với tên **"PREF"**. **Instance** này có thể được sử dụng trong phạm vi toàn bộ ứng dụng bất cứ đâu nó được lấy về.



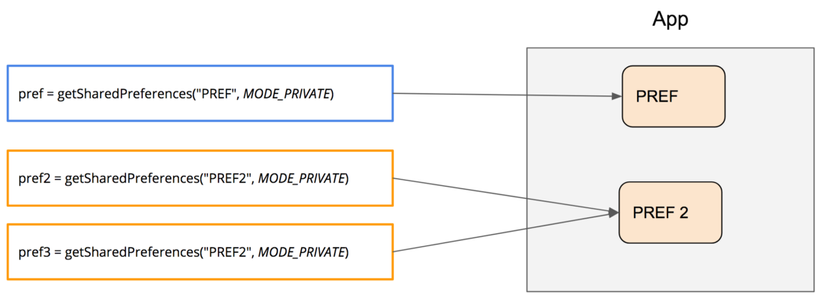


Mỗi một **shared preferences** sẽ có **instance** riêng dựa trên tên của **preference**.



**pref** và **pref2** là 2 **instance** khác nhau bởi vì chúng được lấy từ 2 **Shared Preferences** khác tên.

Không quan trọng trong context nào (Activity, Fragment, Application, ...) mà nó được gọi, nó luôn luôn trả về **instance** giống nhau khi ta gọi method trên với tên giống nhau. Điều này đảm bảo rằng luôn luôn nhận được dữ liệu mới nhất được lưu.



**pref2** và **pref3** sử dụng cùng một tên **Shared Preferences** trong lệnh gọi nên chúng là cùng 1 **instance**

1. **Lấy dữ liệu**

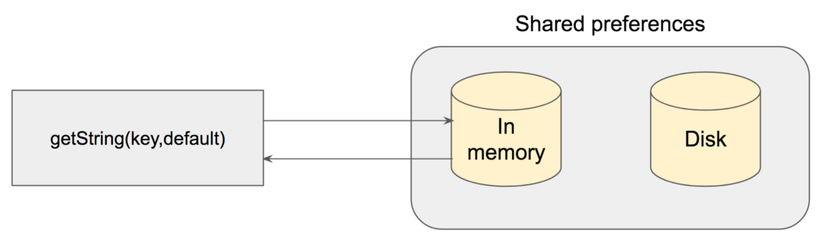
Tất cả các thao tác được đi qua bộ nhớ **In-Memory**, điều đó có nghĩa là ngay lập tức và tránh được các thao tác I/O. Bởi vì tất cả các thao tác đi qua bộ nhớ **in-memory** nên nó đảm bảo rằng giá trị trả về sẽ là giá trị mới nhất.

**Các hàm lấy dữ liệu:**

* getBoolean()
* getInt()
* getLong()
* getFloat()
* getString()

Ví dụ:





**Chú ý:** Nếu lần đầu dữ liệu được tải lên, nó sẽ đợi cho đến khi dữ liệu được loaded vào bộ nhớ **in-memory** từ **disk storage**.

1. **Lưu dữ liệu**

**Các hàm lưu dữ liệu:**

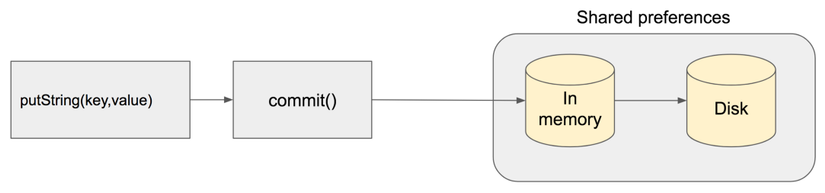
* putBoolean()
* putInt()
* putLong()
* putFloat()
* putString()

**Lưu dữ liệu thông qua commit()**

Lưu dữ liệu vào **in-memory** trước sau đó thực hiện ghi vào disk một cách đồng bộ. Bởi vì có sự tham gia của thao tác I/O, main thread sẽ bị blocked cho đến khi data được ghi hết vào **disk storage**. Đây là một thao tác kém hiệu quả so với **apply()**.

Bởi vì thao tác ghi dữ liệu lên **disk storage** là đồng bộ cho nên trạng thái trả về sẽ là một biến boolean. Do đó chúng ta có thể xác nhận rằng liệu thao tác mà ta đang thực hiện có thành công hay không bằng cách kiểm tra kết quả trả về.





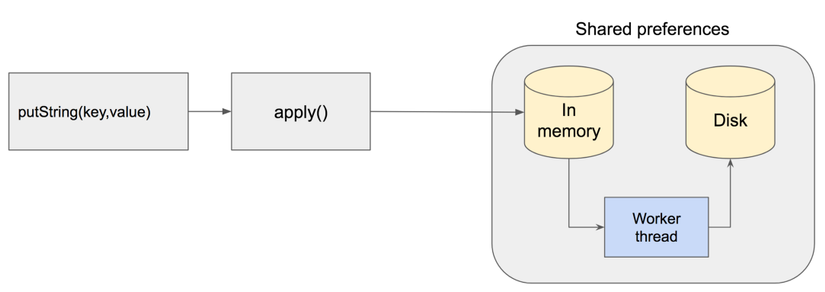
**Lưu giá trị thông qua apply()**

Lưu giá trị vào **in-memory** trước sau đó ghi dữ liệu một cách bất đồng bộ vào **disk-storage**. Main thread sẽ không bị blocked và không phải đợi thao tác trên disk nữa. Tuy thao tác ghi lên **disk storage** là **asynchronous** nhưng bất kỳ thao tác **read** ngay sau lời gọi **apply()** sẽ đều trả về kết quả là mới nhất bởi vì thao tác **read(get)** sẽ được thực thi thông qua **in-memory.**

Bởi vì thao tác ghi dữ liệu lên **disk-storage** là **asynchronous**, sẽ không có giá trị nào được trả về. Do đó bạn không thể xác nhận được là liệu thao tác bạn thực thi có thành công hay không,

Hãy sử dụng **apply** trừ khi cần xác nhận kết quả của thao tác thực hiện.





1. **Các thao tác khác**

Ngoại trừ việc **put** và **get**, **shared preferences** còn cung cấp các thao tác sau:

* Kiểm tra xem một key có tồn tại hay không.
* Lắng nghe việc thay đổi của shared preferences thông qua listener.
* Đọc toàn bộ các entries thành map.
* Xóa bỏ một entry..
* Xóa bỏ toàn bộ các entries (Clear all).

Ngoài ra, **Shared preferences** phạm vi ứng dụng được lưu tại đường dẫn



Có thể lấy nó ra và mở ra xem dữ liệu bên trong.

* 1. **SQLite**

Nếu đang viết một ứng dụng cần duy trì và truy vấn một lượng lớn dữ liệu trên thiết bị cục bộ, hãy cân nhắc sử dụng cơ sở dữ liệu thay vì tệp cục bộ hoặc cửa hàng khóa-giá trị. Nói chung, cơ sở dữ liệu cung cấp chèn nhanh hơn, cập nhật, và truy vấn so với các giải pháp kiên trì khác của địa phương

Các ứng dụng Flutter có thể sử dụng cơ sở dữ liệu SQLite thông qua plugin sqflite có sẵn trên pub.dev. Công thức này trình bày những điều cơ bản của việc sử dụng để chèn, đọc, cập nhật và xóa dữ liệu về các Dogs.sqflite khác nhau.

**Công thức này sử dụng các bước sau:**

* Thêm các phần phụ thuộc.
* Xác định mô hình dữ liệu .Dog
* Mở cơ sở dữ liệu.
* Tạo bảng.dogs
* Chèn .Dog vào cơ sở dữ liệu
* Lấy danh sách chó.
* Cập nhật .Dog trong cơ sở dữ liệu
* Xóa .Dog khỏi cơ sở dữ liệu
  + 1. **Thêm các phụ thuộc**

Để làm việc với cơ sở dữ liệu SQLite, hãy nhập các gói và **.sqflitepath**

- Gói **.sqflite** cung cấp các lớp và hàm để tương tác với cơ sở dữ liệu SQLite

- Gói **.path** cung cấp các chức năng để Xác định vị trí để lưu trữ cơ sở dữ liệu trên đĩa

Để thêm các gói dưới dạng phụ thuộc, chạy **flutter pub add** :

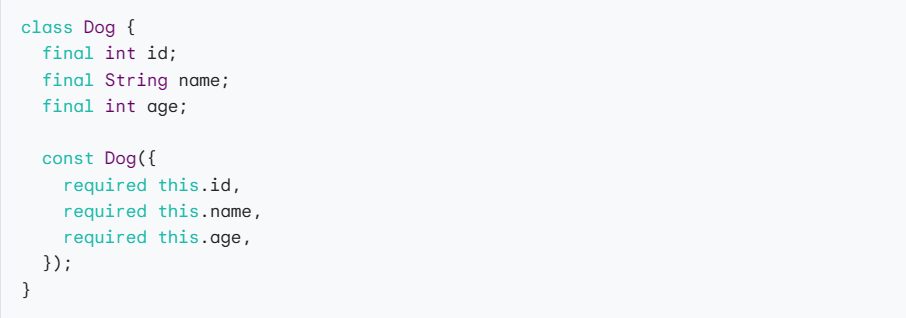


Các gói trong tệp trong một dự án Flutter phải đảm bảo được nhập khi làm việc:



* + 1. **Xác định mô hình dữ liệu Dog**

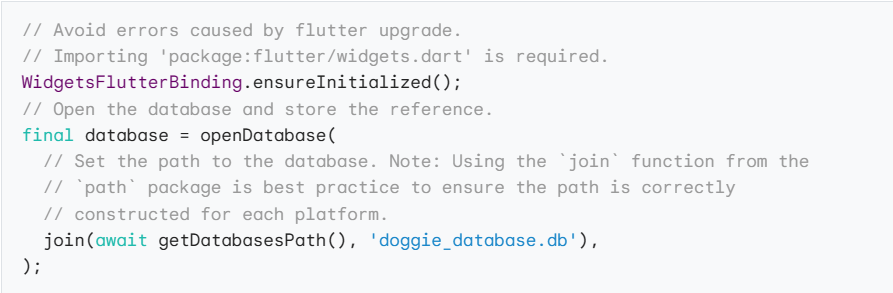
Trước khi tạo bảng để lưu trữ thông tin về Dogs, hãy dành một vài phút để xác định dữ liệu cần được lưu trữ. Trong ví dụ này, hãy định nghĩa một lớp Dog chứa ba phần dữ liệu: **id**, **name**, **age.**



* + 1. **Mở cơ sở dữ liệu**

Trước khi đọc và ghi dữ liệu vào cơ sở dữ liệu, hãy mở một kết nối vào cơ sở dữ liệu. Điều này bao gồm hai bước:

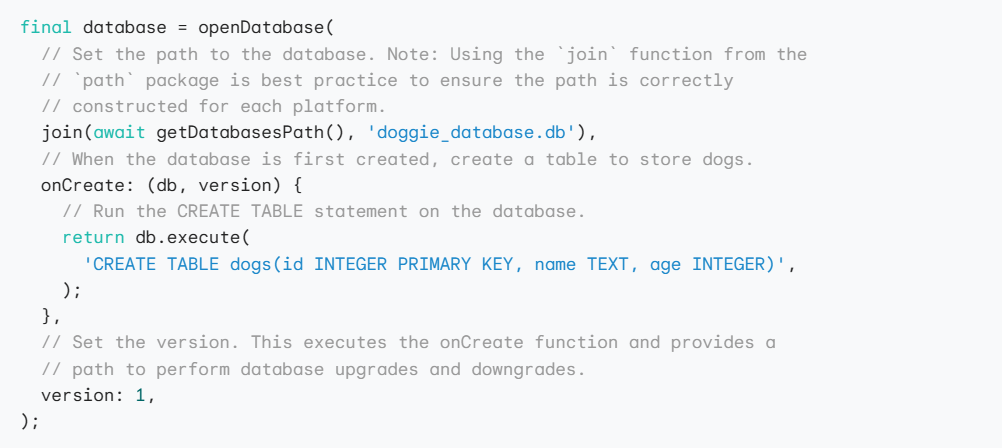
* Xác định đường dẫn đến tệp cơ sở dữ liệu bằng cách sử dụng .getDatabasesPath() từ gói sqflite, kết hợp với join hàm từ gói path
* Mở cơ sở dữ liệu với hàm .openDatabase() từ sqflite



* + 1. **Tạo bảng dogs**

Tiếp theo, tạo một bảng để lưu trữ thông tin về các Dogs khác nhau. Trong ví dụ này, hãy tạo một bảng được gọi là dogs xác định dữ liệu có thể được lưu trữ. Mỗi Dog chứa một id, name, age. Do đó, chúng được biểu diễn dưới dạng ba cột trong bảng dogs.

* id là một Dart int, và được lưu trữ như một INTERGER kiểu dữ liệu của SQLite. Nó cũng là thực hành tốt để sử dụng một id làm chính cho bảng để cải thiện thời gian truy vấn và cập nhật.
* name là một Dart String, và được lưu trữ như một TEXT kiểu dữ liệu của SQLite.
* age cũng là một Dart int , và được lưu trữ dưới dạng INTEGER.

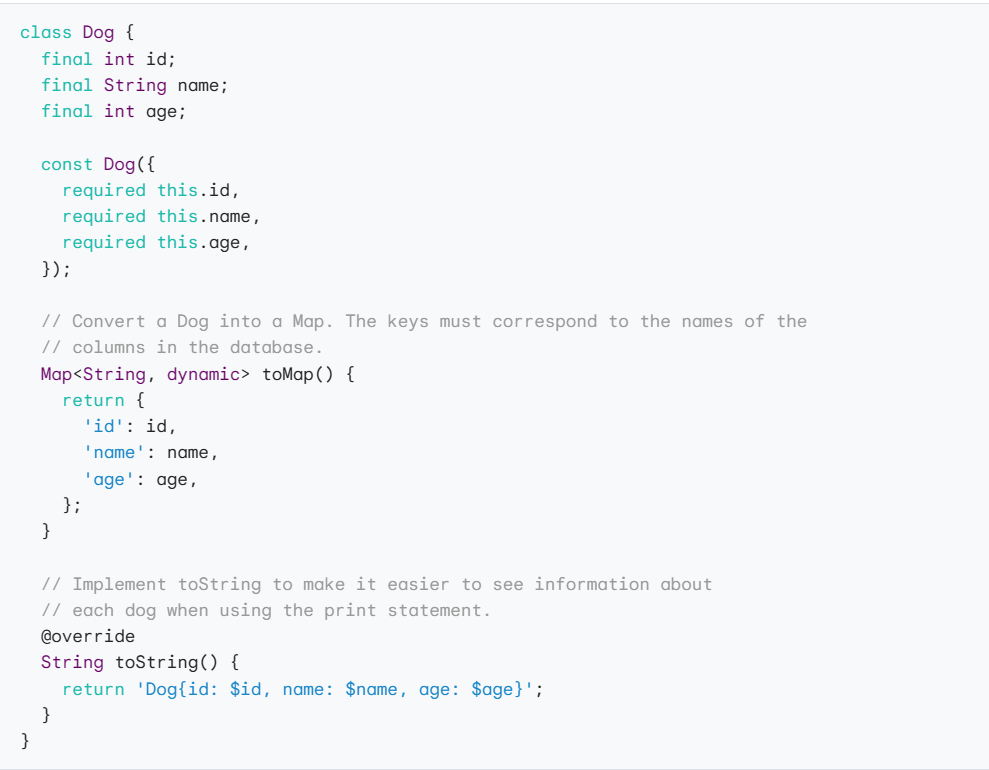


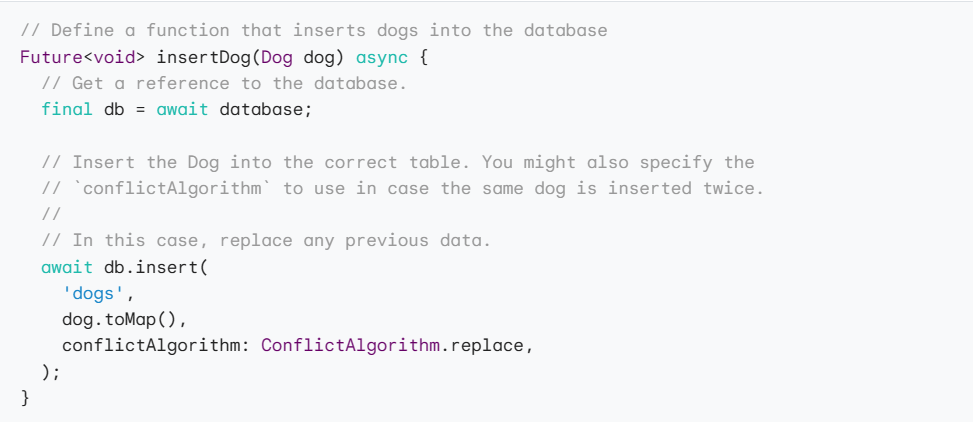
* + 1. **Chèn một vào cơ sở dữ liệu**

Bây giờ đã có một cơ sở dữ liệu với một bảng phù hợp để lưu trữ thông tin về những dogs khác nhau, đã đến lúc đọc và ghi dữ liệu.

Đầu tiên, chèn một Dog vào bảng dog. Điều này bao gồm hai bước:

* Chuyển đổi Dog thành một Map
* Sử dụng phương thức insert() để lưu trữ Maptrong bảng dogs.





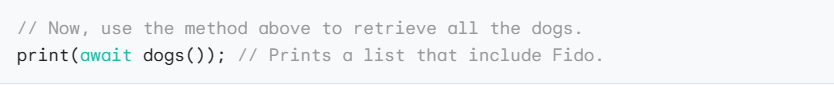


* + 1. **Truy xuất danh sách Dog**

Bây giờ Dog được lưu trữ trong cơ sở dữ liệu, hãy truy vấn cơ sở dữ liệu cho một cụ thể hoặc một danh sách tất cả các. Điều này bao gồm hai bước:

* Chạy query dựa vào bảng dogs. Thao tác này trả về tệp List<Map>.
* Chuyển đổi List<Map> thành tệp List<Dog>.



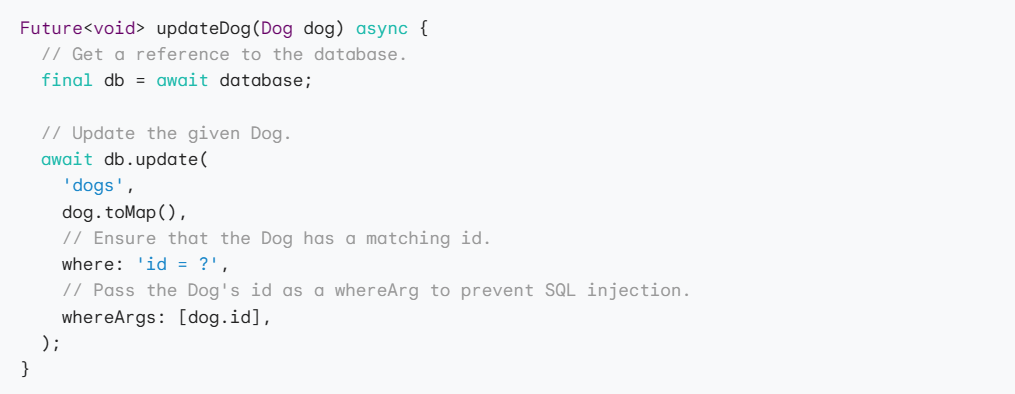


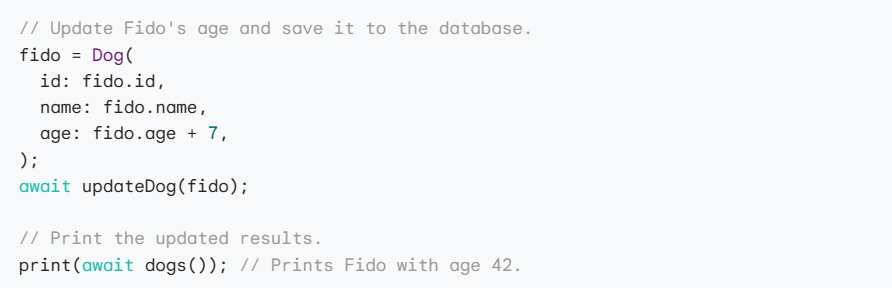
* + 1. **Cập nhật Dog trong cơ sở dữ liệu**

Sau khi chèn thông tin vào cơ sở dữ liệu, chúng ta có thể cập nhật thông tin đó tùy theo nhu cầu sử dụng. Để có thể làm điều này, chúng ta có thể thực hiện bằng cách sử dụng phương thức update() từ thư viện.sqflite

Điều này bao gồm hai bước:

* Chuyển đổi Dog thành bản đồ.
* Sử dụng where một mệnh đề để đảm bảo bạn cập nhật đúng.





* + 1. **Xóa Dog khỏi cơ sở dữ liệu**

Ngoài việc chèn và cập nhật thông tin về Dog, ta cũng có thể xóa dog khỏi cơ sở dữ liệu. Để xóa được dữ liệu một cách dễ dàng và nhanh chóng, có thể sử dụng phương thức delete() từ thư viện.sqflite

Trong phần này, tạo một hàm lấy id và xóa bằng ID phù hợp từ cơ sở dữ liệu. Để thực hiện công việc này, chúng ta cần phải cung cấp một where điều khoản để giới hạn các bản ghi bị xóa.



1. **3rd Party Librabies flutter**

“3rd Party Librabies Flutter” là các thư viện được gọi là gói mã nguồn được tạo ra bởi các nhà phát triển khác, không nằm trong phần Flutter Framework chính. Tuy nhiên, chúng vẫn được sử dụng để cung cấp các tính năng bổ sung, công cụ, hoặc đưa ra giải pháp để phát triển ứng dụng Flutter một cách nhanh chóng và đạt được hiệu quả cao.

Ngoài ra, các thư viện ở bên thứ ba có thể cung cấp các chức năng như việc quản lý trạng thái ứng dụng, giao diện người dùng hay gọi các API của mạng,…

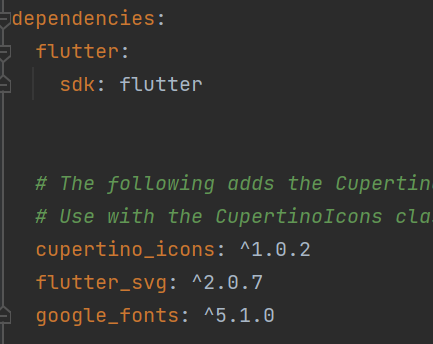
Sau đây sẽ là một số thư viện hay được sử dụng trong một dự án của Flutter:

* Flutter\_SVG

**“flutter\_svg”** cho phép hiển thị hình ảnh SVG (Scalable Vector Graphics) trong ứng dụng Flutter. Thư viện này giúp cho chúng ta có thể tận dụng nhiều nhất có thể các ưu điểm của ảnh vector, như phóng to hoặc thu nhỏ mà không làm mất đi độ nét, tạo hiệu ứng hoặc tùy chỉnh tùy theo nhu cầu sử dụng trong dự án Flutter của mình.

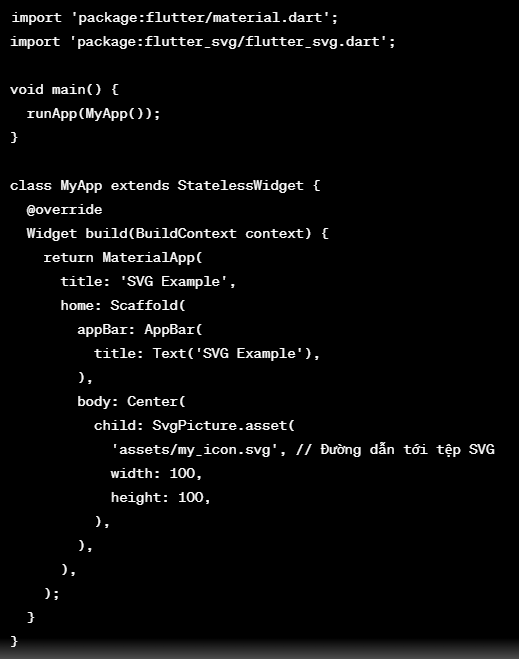
Cách để sử dụng **“flutter\_svg”:**

Trong file , ta sẽ thêm thư viện “flutter\_svg” vào như sau



Sau đó ta sẽ chạy lệnh  để cài đặt thư viện vào trong dự án.

Ngoài ra, ta cũng có thể sử dụng flutter\_svg trong mã nguồn như này:

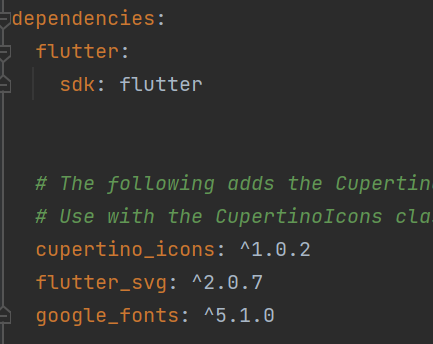


* Google fonts

**“google\_fonts”** là thư viện cho phép dễ dàng sử dụng các phông chữ từ bộ fonts của Google trong dự án Flutter. Nó sẽ cung cấp tính năng tích hợp các phông chữ Google vào giao diện người dùng của dự án đang thực thi mà không cần phải mất thời gian trong việc tải về thủ công các tệp fonts của Google.

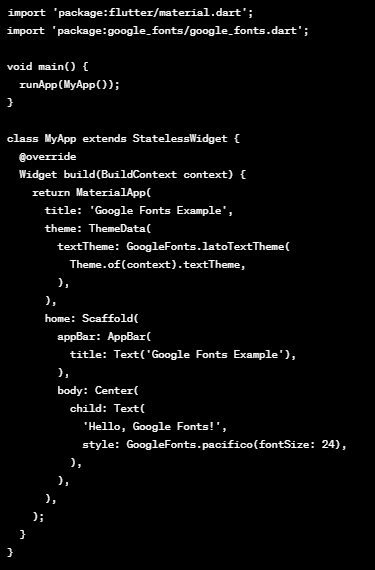
Cách để sử dụng **“google\_fonts”:**

Trong file , ta sẽ thêm thư viện **“google\_fonts”** vào như sau



Sau đó ta sẽ chạy lệnh  để cài đặt thư viện vào trong dự án.

Ngoài ra, ta cũng có thể sử dụng **google\_fonts** trong mã nguồn như này:

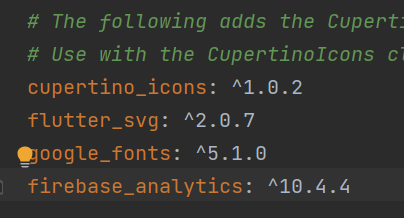


* Firebase analytics

“firebase\_analytics” là thư viện được sử dụng để tích hợp dịch vụ phân tích từ Firebase vào dự án Flutter. Nó cung cấp các dịch vụ dùng cho việc phân tích dữ liệu để chúng ta có thể theo dõi và hiểu hành vi của người dùng trong ứng dụng flutter đang phát triển.

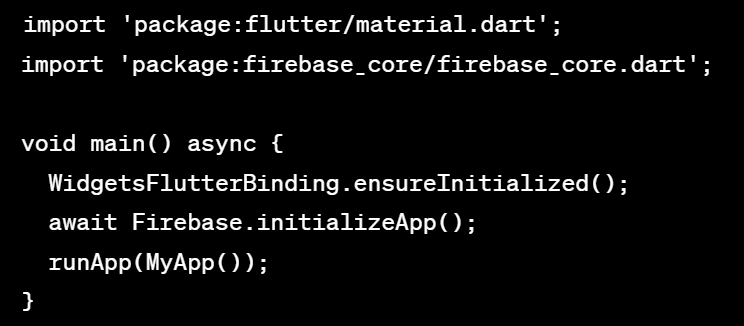
Cách để sử dụng **“firebase\_analytics”:**

Trong file , ta sẽ thêm thư viện **“firebase\_analytics”** vào như sau



Sau đó ta sẽ chạy lệnh  để cài đặt thư viện vào trong dự án.

Ngoài ra, ta cũng có thể sử dụng firebase\_analytics trong mã nguồn như này:



1. **Behavior Components**

**“Bahavior Components”** là những thành phần, thư viện hoặc những đối tượng liên quan đến công tác quản lý hành vi, tương tác lẫn luồng làm việc trong flutter.

Một số ví dụ về thu viện Beahvior Components:

**3.1. State Management Libraries (Thư viện quản lý trạng thái):**

* **Provider:** Thư viện quản lý trạng thái dựa trên InheritedWidget, giúp chia sẻ trạng thái và cập nhật UI một cách hiệu quả.
* **Bloc:** Sử dụng mô hình BLoC để quản lý trạng thái và luồng dữ liệu giữa giao diện người dùng và logic kinh doanh.
* **GetX:** Thư viện cung cấp cách dễ dàng quản lý trạng thái, dependency injection và routing.
  1. **Navigation Libraries (Thư viện điều hướng):**
* **Navigator:** Flutter tích hợp Navigator để quản lý luồng điều hướng giữa các màn hình.
* **PageRouteBuilder:** Cho phép tạo ra các hành động chuyển đổi màn hình tùy chỉnh.
* **RxDart:** Một thư viện mở rộng cho ReactiveX (Rx) để quản lý và xử lý luồng dữ liệu.

**3.3. Gesture Detection (Phát hiện cử chỉ):**

* **GestureDetector**: Cho phép phát hiện và xử lý các cử chỉ như chạm, vuốt, nhấn và giữ.
* **InkWell:** Một widget tương tác cho phép thêm hiệu ứng "splash" khi người dùng nhấn vào.
* **Draggable và DraggableScrollableSheet**: Cho phép thực hiện kéo thả và cuộn có thể kéo.

**3.4. Animation Libraries (Thư viện hoạt ảnh):**

* **AnimationController**: Cho phép điều khiển các hoạt ảnh và tạo hiệu ứng chuyển động.
* **Tween và TweenAnimationBuilder:** Sử dụng để tạo các hoạt ảnh dựa trên giá trị trung gian.
* **Hero:** Để thực hiện chuyển đổi hoạt ảnh mượt mà giữa các màn hình.

**3.5. User Interaction Libraries (Thư viện tương tác người dùng):**

* **AlertDialog:** Để hiển thị hộp thoại thông báo và xác nhận.
* **SnackBar:** Hiển thị thông báo ngắn thời gian.
* **BottomSheet:** Hiển thị các hình thức chọn và tương tác phía dưới màn hình.

**3.6. Routing and Navigation (Điều hướng và định tuyến):**

* **Navigator:** Quản lý luồng điều hướng giữa các màn hình trong ứng dụng.
* **PageRouteBuilder:** Tạo các hành động chuyển đổi màn hình tùy chỉnh.
* **Routes và MaterialApp:** Định tuyến và quản lý các màn hình ứng dụng.

Các **Behavior Components** đóng một vai trò quan trọng trong việc xây dựng các ứng dụng phức tạp và tương tác giữa người dùng với nhau. Một số ứng dụng có thể kể đến như ứng dụng mạng xã hội, sàn giao dịch thương mại điện tử, liên lạc trực tuyến… Cụ thể như là:

* **State Management:** Quản lý tiến trạng thái người dùng, bình luận, lượt thích, giỏ hàng, tình trạng giỏ hàng…
* **Event Handling:** Xử lý các hành động tương tác cụ thể như tương tác cảm xúc trên ứng dụng xã hội, đăng nhập/đăng ký tài khoản, nộp bài…
* Navigation: Điều hướng giữa các trang cá nhân, danh mục sản phẩm, giữa các bài học, bài kiểm tra, trang web liên quan…
* **Animation:** Hiển thị hiệu ứng hoạt ảnh khi mỗi khi người dùng thực hiện tương tác với ứng dụng như thêm sản phẩm, thả cảm xúc bài viết…
* **User Interaction:** Hiển thị thông báo, tin tức liên quan đến người dùng như thông tin đơn hàng, trao đổi giữa người bán với người mua, khuyến mãi, thông tin bài kiểm tra…
* **Data Flow:** Luồng dữ liệu giữa những người dùng với nhau hoặc từ người dùng với quản trị viên. Những dữ liệu này bao gồm những nội dung liên quan như thông tin đơn hàng, thông tin sản phẩm, kết quả kiểm tra…

1. **Sate Managements**

**4.1. Quản lý trạng thái là gì?**

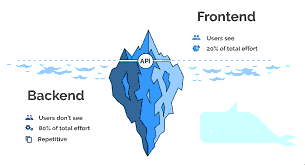
Quản lý trạng thái ứng dụng là quá trình duy trì kiến thức về đầu vào của ứng dụng trên nhiều luồng dữ liệu liên quan tạo thành một giao dịch kinh doanh hoàn chỉnh hoặc một phiên để hiểu tình trạng của ứng dụng tại bất kỳ thời điểm nào. Trong khoa học máy tính, đầu vào là thông tin được người dùng đưa vào chương trình và trạng thái đề cập đến tình trạng của một ứng dụng theo đầu vào được lưu trữ của nó được lưu dưới dạng biến hoặc hằng số.

Các ứng dụng kinh doanh cốt lõi xử lý thông tin nhạy cảm như đơn đặt hàng, thanh toán, hóa đơn và hóa đơn vật liệu. Việc xử lý này phụ thuộc vào trạng thái của ứng dụng và dẫn đến thay đổi số dư hàng tồn kho, tài khoản và sổ tài chính. Quản lý trạng thái cho phép các nhà phát triển xác định trạng thái của ứng dụng để đảm bảo các thay đổi được thực hiện phản ánh phù hợp bối cảnh và quy trình kinh doanh trong thế giới thực.

* 1. **Quản lý trạng thái hoạt động như thế nào ?**

Quản lý trạng thái làm cho trạng thái của một ứng dụng hiển thị dưới dạng cấu trúc dữ liệu, cải thiện khả năng làm việc với ứng dụng của nhà phát triển. Thư viện quản lý trạng thái cung cấp cho các nhà phát triển các công cụ cần thiết để tạo cấu trúc dữ liệu và thay đổi chúng khi các hành động mới xảy ra.

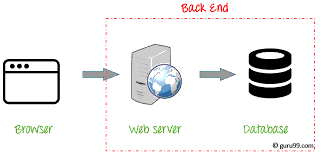
Có hai mô hình được chấp nhận để quản lý trạng thái: Front end (còn gọi là phía máy khách) và back end (đôi khi được gọi là phía máy chủ).



Trong quản lý trạng thái front-end, ứng dụng hoặc trình duyệt của chính người dùng duy trì trạng thái của chương trình, thường bằng cách bật hoặc tắt một số nút hoặc tính năng giao diện người dùng nhất định và gửi trạng thái cùng với tin nhắn. Kết nối giao diện người dùng đảm bảo người dùng và ứng dụng hài hòa trong suốt phiên.



Trong quản lý trạng thái back-end, một thành phần ứng dụng sẽ sử dụng cấu trúc dữ liệu ngoài hoặc cơ sở dữ liệu để ghi lại trạng thái cuối cùng khi thực hiện các hoạt động xử lý. Việc xử lý thư tiếp theo sẽ bắt đầu bằng việc truy xuất trạng thái trước đó từ cơ sở dữ liệu. Biến trạng thái trong cấu trúc dữ liệu cũng có thể đồng bộ hóa giao diện người dùng và thông qua nó là người dùng với trạng thái của phiên.



* 1. **Thư viện quản lý trạng thái**

Bất kỳ việc thực hiện quản lý trạng thái nào cũng có khả năng được hưởng sự hỗ trợ của các công cụ phát triển được thu thập trong thư viện quản lý trạng thái. Các thư viện này đảm bảo việc triển khai quản lý trạng thái front-end hoặc back-end làm việc cùng nhau để có được thành công và cung cấp sự hiểu biết và kiểm soát ứng dụng.

Thư viện quản lý trạng thái tăng tốc độ phát triển trạng thái ứng dụng và làm cho mã quản lý trạng thái dễ duy trì hơn bằng cách thực thi các thông lệ phổ biến trên nhiều ứng dụng trạng thái. Điều quan trọng là chọn các kỹ thuật quản lý trạng thái phù hợp với ứng dụng, cũng như thư viện công cụ tốt nhất để thực hiện các kỹ thuật này.

Thư viện quản lý trạng thái thể hiện rõ các kỹ thuật phát triển như ngôn ngữ phát triển hoặc bản chất front-end hoặc back-end của kiểm soát trạng thái. Điều quan trọng là các nhóm phát triển phải nghiên cứu các lựa chọn của họ và đưa ra lựa chọn tốt nhất cho cả nhu cầu hiện tại và có thể trong tương lai.

**Ta có những thư viện quản lý trạng thái sau :**

* Usability
* Performance
* Scalability
* Modifiability
* Maintainability
* Reusability
* Testability
* Ecosystem
* Community



* 1. **Quản lý trạng thái trong phát triển ứng dụng .**

Các ứng dụng ban đầu kiểm soát hộp thoại người dùng, vì vậy các bước riêng lẻ trong một phiên được quyết định bởi chính quy trình :

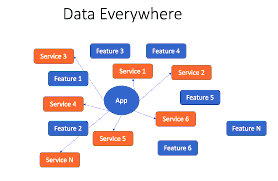
* Ghi dữ liệu cho người dùng.
* Chấp nhận thay đổi.
* Xác nhận chúng và cam kết chúng.

Với sự ra đời của các ứng dụng web dựa trên tầm nhìn không xác định của HTML, việc xác định rõ ràng hơn các hoạt động quản lý trạng thái trở nên cần thiết.

Khi nhận được thông báo phiên chẳng hạn như trang HTML nó được diễn giải dựa trên trạng thái, được ghi lại dưới dạng biến hoặc hằng số mà mọi quy trình đều có thể truy cập. Bản ghi đảm bảo rằng việc nhận, ví dụ, thông báo "xác nhận" trong khi ở trạng thái "chờ xác nhận" được xử lý khác với trạng thái "chờ quá lâu cho trạng thái phản hồi của người dùng. Đến lượt nó, đảm bảo rằng các hệ thống ứng dụng cập nhật cơ sở dữ liệu và tạo ra hồ sơ kinh doanh bền vững luôn được đồng bộ hóa với thế giới thực.

* 1. **Ưu điểm của quản lý trạng thái .**

Quản lý trạng thái là điều cần thiết trong việc điều chỉnh và tích hợp các ứng dụng kinh doanh cốt lõi và đám mây. Nếu không có một số hình thức quản lý trạng thái, các hoạt động kinh doanh thường xuyên như mua một cái gì đó hoặc yêu cầu cung cấp thông tin sẽ phải được cấu trúc như một yêu cầu hoặc trao đổi phản hồi duy nhất. Điều này có thể gây gánh nặng đáng kể cho người dùng và gần như chắc chắn sẽ làm giảm hiệu quả của ứng dụng. Trong một số trường hợp, chẳng hạn như việc xử lý đơn đặt hàng, một sàn giao dịch không trạng thái có thể che giấu thông tin quan trọng như mức tồn kho hiện tại, dẫn đến những gì có thể là tác động kinh doanh đáng kể đến người bán và gây bất tiện lớn cho người mua.



* 1. **Công cụ .**

Các công cụ quản lý trạng thái thường được cung cấp dưới dạng thư viện quản lý trạng thái, được thiết kế cho các nhà phát triển muốn xây dựng nhận thức về trạng thái vào các ứng dụng của họ. Hầu hết các công cụ này được sử dụng để thực hiện quản lý trạng thái front-end vì cơ chế đó dễ thực thi nhất và cung cấp sự tích hợp chặt chẽ nhất với cơ sở người dùng của ứng dụng. Kiểm soát trạng thái front-end cũng lý tưởng cho quản lý trạng thái khi mục tiêu phát triển là thêm giao diện người dùng đám mây vào quy trình kinh doanh.

Hầu hết các công cụ hoạt động bằng cách thêm một đối tượng trạng thái vào một cái gì đó như JavaScript. Đối tượng này và các phương thức của nó sau đó được sử dụng để quản lý trạng thái và ngăn ngừa mất trạng thái trong các điều kiện bất thường, bao gồm lây nhiễm bởi mã hoặc thay đổi mối quan hệ máy khách và máy chủ. Các thư viện như React tập trung gần như hoàn toàn vào giao diện người dùng. Các thư viện khác, như Redux hoặc Cerebral, cung cấp một hình thức quản lý trạng thái tích hợp được sử dụng trong suốt quy trình làm việc.

Flutter là một SDK có thể được sử dụng rộng rãi hơn một chút, trong cả ứng dụng quản lý trạng thái front-end và back-end.

**Những công cụ trong quản lý trạng thái :**

**RXJS Subjects**: trong một số trường hợp khác, đặc biệt là trên các ứng dụng có độ phức tạp thấp và trung bình trong Angular, chúng ta có thể chọn sử dụng Đối tượng để xử lý dữ liệu và truyền thông tin đến các thành phần của chúng ta một cách tổng quát hơn.

**Redux:** Được khuyến nghị cho các ứng dụng vừa và lớn, đây là giải pháp chung cho bất kỳ thư viện / khung công tác nào mà chúng tôi có thể đang sử dụng trong giao diện người dùng.

**AKITA:** nó tương tự như Redux, nhưng dễ sử dụng hơn bằng cách tránh các hành động và điều phối của chúng ta.

**Mobx:** nó có các chức năng tương tự như Redux, nhưng một lần nữa dễ sử dụng hơn. Nó được khuyến khích cho các dự án nhỏ sử dụng React hoặc Vue.

**Angular**

* NGRX: Ứng dụng Redux hướng đến Angular
* NGXS: tương tự như NGRX nhưng sử dụng cấu trúc và cách làm việc gần với Angular.

**React**

* Context: Nó được cung cấp bởi React trong các phiên bản mới nhất, điều đó sẽ tạo điều kiện cho chúng ta không sử dụng Redux trong các ứng dụng đơn giản.

**Vue**

* Vuex: ứng dụng của Redux với Vue

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**

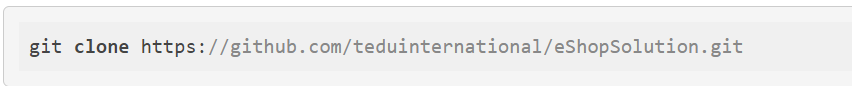
Lệnh này dùng để khởi tạo một Repository trên máy local, tuy nhiên nó chưa được push lên server mà chúng ta vẫn có thể thao tác với tất cả các lệnh như bình thường và nó hoạt động như một repository thực thụ trừ share code cho các thành viên khác.



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.

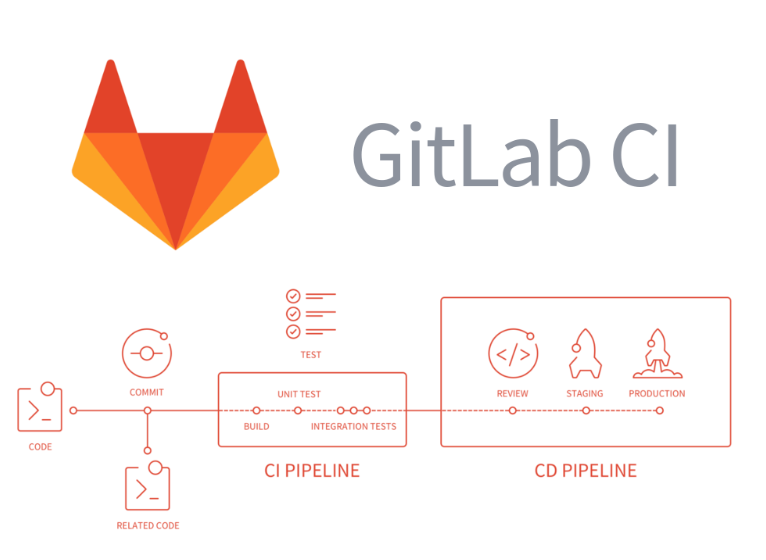
GitHub có Giao diện người dùng đơn giản giúp các nhà phát triển nhanh chóng bắt tay vào làm việc với các thuật toán Git. Một đặc điểm xác định khác là tốc độ của nó - dịch vụ đẩy và kéo các yêu cầu và hợp nhất các phiên bản một cách nhanh chóng.

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.

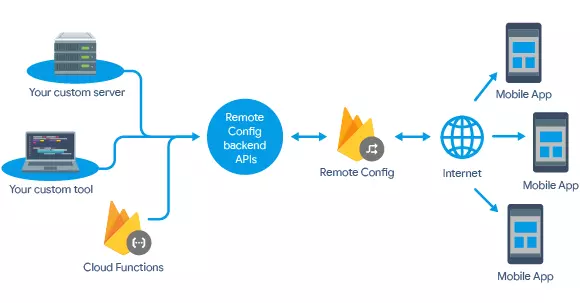


* **Remote Config**

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.

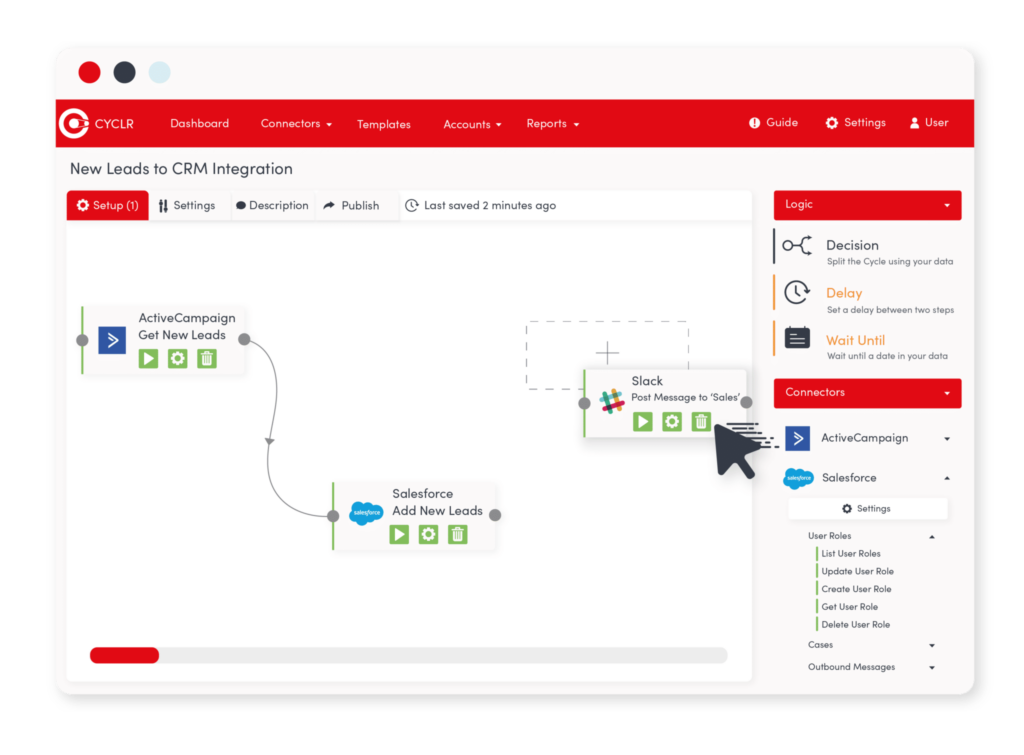
Ví dụ, trong quá trình phát triển ứng dụng di động, tích hợp "native" có thể bao gồm:

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.