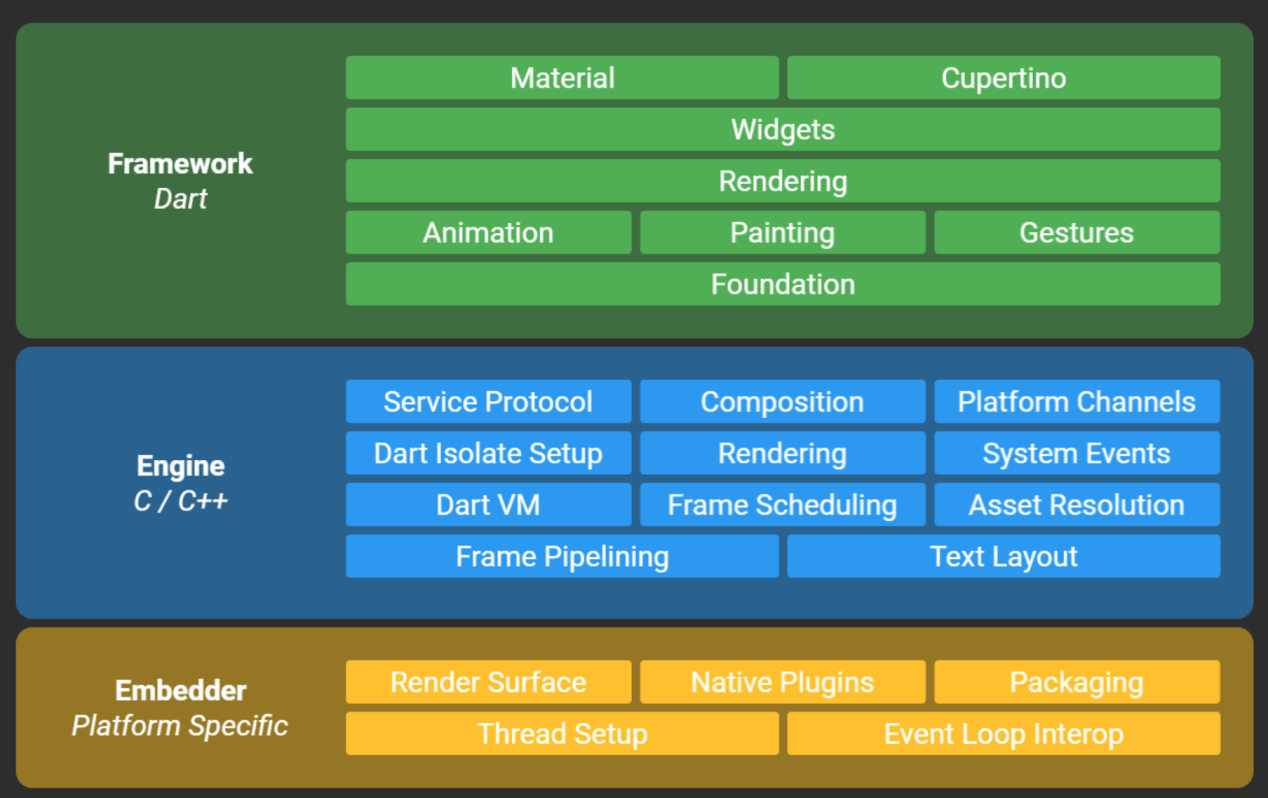
# Flutter

**<https://api.flutter.dev/flutter/foundation/CachingIterable/map.html>**

1. **Flutter Architecture:**

****

1. **When we don’t use Flutter:**

- Game and Audio:

+ sử dụng các nền tảng chuyên cho game sẽ tốt hơn : unity, unreal,...

+ Flutter chưa có nhiều support for audio, nên nếu dùng flutter hiệu năng sẽ ko tốt và khó implement.

- Apps with specific native SDK needs:

+ Điều hiển nhiên. Khi bạn cần ứng dụng tương tác đặc biệt với thiết bị. Thì các nền tảng native là số 1. Và tương lai có ra sao nữa thì với Swift, Kotlin vẫn là những tượng đài không thể sụp đổ được.

+ Ví dụ: ứng dụng của bạn cần tương tác với hệ sinh thái của Apple với CloudKit, native hardware, MLKit, StoreKit … thì SwiftUI sẽ là lựa chọn tốt hơn rất nhiều.

- Certain platform: (nền tảng nhất định):

+ Flutter không chạy ở khắp mọi nơi. Nó chưa hỗ trợ Apple Bitcode, có nghĩa là nó không hỗ trợ watchOS, tvOS hoặc một số tiện ích mở rộng ứng dụng iOS nhất định.

+ Hỗ trợ của nó cho web vẫn là beta hoặc có thể là mới release, điều đó có nghĩa là Flutter có nhiều tính năng và cải tiến hiệu suất trước nó, nhưng chúng sẽ sắp ra mắt.

1. **Flavor Configuration:**

<https://fxstudio.dev/flavor-configurations-flutter/>

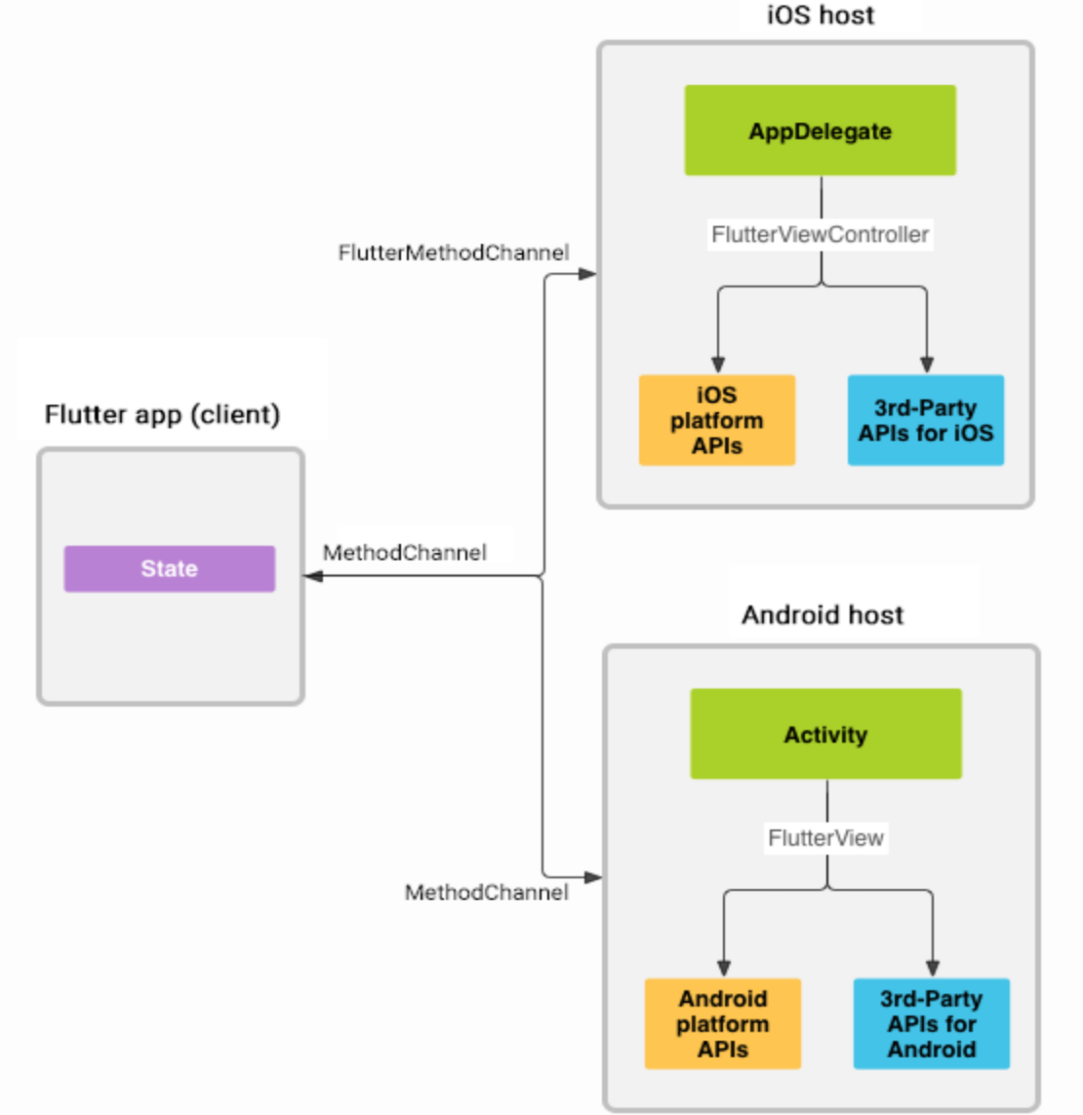
- trong một dự án, bạn cần phải có các môi trường khác nhau để build các app khác nhau phục vụ cho mỗi mục đích riêng biệt: local enviroment, Staging/QA enviroment, Product enviroment.

- Cách cấu hình cho android và IOS là khác nhau.

1. **Method Channel:**

**<https://fxstudio.dev/platform-specific-code-with-methodchannel-flutter/>**

Flutter cho phép 1 base code có thể build được cho nhiều nền tảng. Tuy nhiên, trong lập trình, đôi khi bạn sẽ gặp phải tình huống cần truy cập API dành riêng cho nền tảng bằng ngôn ngữ hoạt động trực tiếp với các API đó (native code). MethodChannel sẽ làm điều đó.



- MethodChannel sẽ được viết, cấu hình ở cả Android (MethodChannel), IOS (FlutterMethodChannel) và Flutter(MethodChannel). Trên Flutter sẽ dùng MethodChannel để invokeMethod gọi đến các funtion được dùng trong MethodChannel của android và IOS.

- MethodChannel được dùng để đọc flavor của mỗi platform.

1. **Run and Debug in Flutter:**

<https://fxstudio.dev/customize-run-and-debug-flutter/>

- VSCode giúp bạn đơn giản hoá việc khởi chạy và debug các cấu hình custom của app. Để làm được điều đó, VS Code sẽ đọc file launch.json nằm trong folder .vscode và việc chúng ta cần làm là viết các cấu hình custom chúng ta muốn vào đấy.

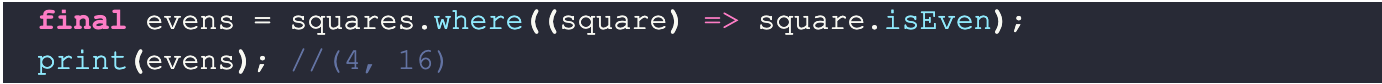
- launch configuration không phải là tính năng của Dart hay Flutter. Nó là một cấu hình để mở rộng khả năng của VS Code.

1. **Collecting in Dart:**

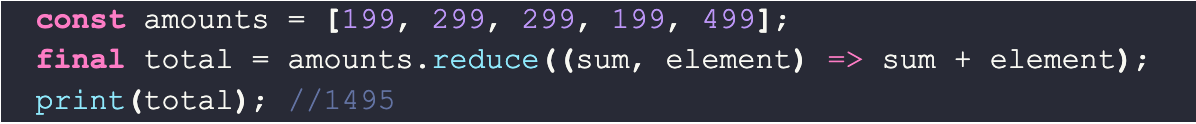
<https://fxstudio.dev/handling-collections-dart-tour/>

- **mapping**: la phép ánh xạ cho một collections sẽ giúp bạn thực hiện một hành động trên từng phần tử của tập hợp, hiểu nôm na như là một vòng lặp. Phương thức map là một anonymous function và kết quả return lại là một collections.

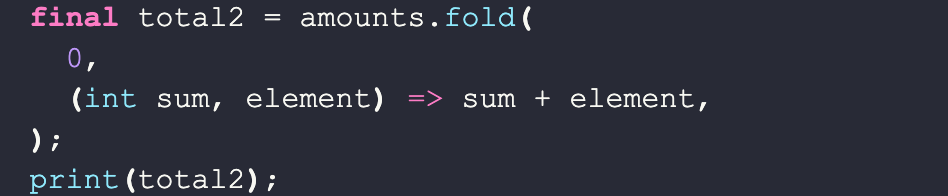
- *Filtering*: lọc một tập hợp để thu lại được một tập hợp nhỏ hơn.



- *Reduce*: là toán tử hợp nhất. Khi bạn muốn tính toán dựa trên toàn bộ số phần tử của tập hợp.

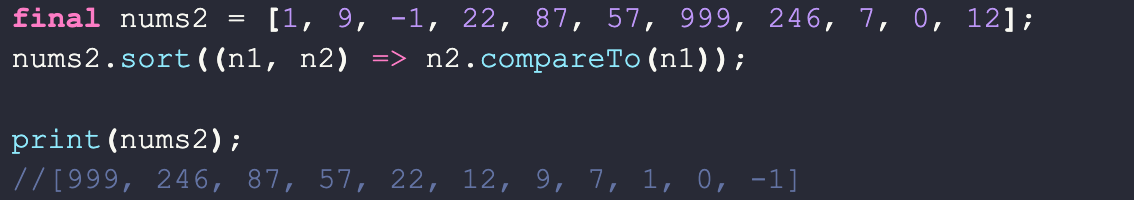


- *Fold*: is same Reduce, but it is more optimize. Reduce trong trường hợp danh sách rỗng thì bạn sẽ gặp lỗi. Và fold sẽ bắt bạn cung cấp giá trị ban đầu khi thực hiện công việc hợp nhất này.



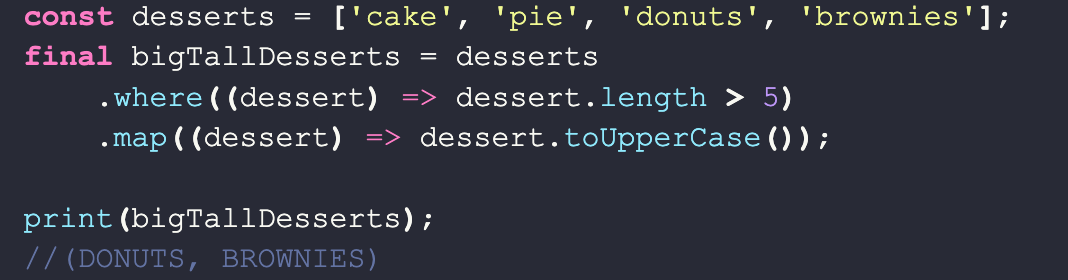
- *Sorting*: Quy luật sắp xếp sẽ dựa trên kiểu dữ liệu của tập hợp mà quyết định. Quan trọng là việc sắp xếp vẫn thực hiện trên chính tập hợp đó, nó sẽ thay đổi thứ tự các phần tử. Và nếu bạn áp dụng chọn const thì sẽ gặp lỗi.

- Custom sorting: áp dụng một quy luật sắp xếp khác.



- Reversing: đảo ngược lại danh sách.

- Combining: có thể kết hợp nhiều toán tử the higher order methods với nhau.



1. **Object Orient Programing: (OOP)**

<https://fxstudio.dev/object-oriented-programming-oop-dart-tour/>

<https://viblo.asia/p/dartflutter-so-sanh-abstract-class-interface-mixin-MG24BrX54z3>

- *Kế thừa*: dùng extends.

- *Overrding*: Lớp con sẽ ghi đè lại các phương thức của lớp cha, từ khoá @override để thực hiện việc ghi đè.

- *Abstract class*:

+ Lớp này không tạo được đối tượng của chính nó, bạn sẽ thấy vắng mặt đi các hàm khởi tạo

+ Nó sẽ không kế thừa từ một lớp nào khác.

+ Các phương thức trong lớp trừu tượng thì thiếu đi dấu {}. Và bạn không cần thiết phải triển khai một phương thức nào đó.

+ Lớp trừu tượng thì không có nghĩa các thuộc tính & phương thức của nó không được triển khai. Vì chúng nó vẫn kế thừa từ Object (thuỷ tổ)

+ subclass của abstract class sẽ implement cac phuong thuc cua abstract class.

- *Interfaces*:

+ Với Dart, mặc định mọi lớp đều là interface, không có từ khoá chuyên biệt để khai báo một Interface.

+ nên dùng abstract để khai báo 1 Interface.

+ sử dụng từ khoá là implements. Để triển khai một interface vào trong một class bất kì. Cách này tương tự với thừa kế.

+ Khi một lớp được coi là giao diện (interface), thì lớp triển khai (implement) nó phải định nghĩa lại mọi phương thức, thuộc tính có trong interface.

- *Phân biệt Extends và Implement*:

+ là 2 cách để mở rộng một lớp : thừa kế một lớp khác và implement một interface.

+ extends thì bạn có quyền truy cập vào tất cả các thuộc tính và phương thức của lớp đó. Mà không cần phải override lại chúng.

+ implement thì bạn sẽ phải định nghĩa lại override các phương thức và thuộc tính ở lớp đó, trong lớp của mình. Và chúng có ý nghĩa trong phạm vi lớp implement mà thôi.

+ không thể extends 2 class trở lên. Tất nhiên 1 con không thể có 2 cha.

+ có thể implement nhiều interfaces vào cho 1 class.

Screen Shot 2023-05-21 at 6.00.03 PM

+ Kết hợp được cả extends và implements.

Screen Shot 2023-05-21 at 6.00.03 PM

- *MIXINs*:

+ is like protocol/delegate in ios.

+ là một cách để sử dụng lại các phương thức hoặc biến giữa các lớp không liên quan.

+ Để sử dụng *Mixin* vào lớp trong OOP, thì bạn sẽ dùng từ khoá *with*.

- *Extension*:

+ thêm các phương thức riêng của mình vào một lớp có sẵn nào đó, mà không cần phải kế thừa lại chúng.

1. **Asynchronous Program:**

- Dart là ngôn ngữ single-thread. Không có nghĩa nó có 1 thread duy nhất trong vùng the isolate. Và cũng không có nghĩa là bạn không chạy tác vụ trên một thread khác được

- Parallelism: thực hiện nhiều task cùng một thời điểm trên bộ vi đa xử lý hoặc nhiều core (nhân) của CPU.

- Concurrency: là đồng thời xử lý nhiều task thay phiên nhau trên cùng 1 core CPU.

- Parallelism vẫn tốt hơn, khi chúng tối ưu được phần cứng và các task vụ thực hiện độc lập với nhau.

Tuy nhiên, có ưu thì phải có nhược và Parallelism cũng không tránh khỏi. Vấn đề xảy ra khi nhiều luồng cùng tác động và một dữ liệu, thay đổi chúng … hệ quả dẫn tới nhiều vấn đề đau thương hơn bạn nghĩ.

* 1. **Isolate**:

- Với đơn luồng của Dart, những gì được chạy trong đó thì gọi là isolate.

- Trong dart chỉ có 1 mainThread sẽ không tạo ra được các thread khác, thay vào đó chúng ta sẽ tạo ra các Isolate.

- Mỗi isolate trong Dart, thì được cấp phát vùng nhớ riêng biệt, nhằm không bị ảnh hưởng qua lại với nhau. Nên để tương tác với nhau thì chúng phải gửi các message: SendPort , ReceiverPort.

- Nhược điểm: chương trình sẽ chạy chậm.

* 1. **Event loop:**

- là quản lý các thread để dart có thể thực hiện concurrency.

- Event loop có 2 queue:

microtask queue: là hàng đợi được dùng nhiều nhất.

event queue: dùng để thu thập sự kiện người dùng.

- Sự hoạt động trong Dart như sau:

+ Synchronous tasks trong main thì luôn luôn phải được chạy ngay, bạn không thể dùng chúng được

+ long-running tasks thì bạn có thể dừng và chuyển nó qua event queue

+ Khi synchronous tasks chạy xong thì Dart sẽ kiểm tra microtask. Nếu còn task thì sẽ đưa vào main thread để chạy tiếp. Nó sẽ lặp lại khi microtask hết task.

+Cả 2 synchronous tasks & microtask đều hết rồi, thì event loop sẽ dừng chờ.

+Khi có một task mới vào microtask thì nó sẽ chờ các task trước đó thực thi xong rồi mới tới lượt mình chạy.

+Quá trình này sẽ thực thi tới khi nào tất cả các queue đều rỗng.

* 1. **Furture:**

- là một kiểu dữ liệu, sẽ trả lại giá trị cho bạn trong tương lai.

- sử dụng Future cho các tác vụ mà bạn không biết chắc được thời gian chúng hoàn thành. Hoặc các tác vụ bất đồng bộ.

- Future là kết quả của hoạt động bất đồng bộ và có 2 trạng thái là chưa hoàn thành và hoàn thành:

+ Chưa hoàn thành: khi chúng ta gọi một hoạt động bất đồng bộ, nó sẽ trả về một Future chưa hoàn thành, đây trạng thái của Future trước khi trả về kết quả.

+ Hoàn thành: khi hoạt động bất đồng bộ thực hiện xong thì Future sẽ ở trạng thái hoàn thành, Future có thể hoàn thành với một giá trị hoặc là một lỗi.

* 1. **Callback:**

- dùng để lấy giá trị trong một Future.

- then để lấy giá trị khi Future kết thúc, nó có kiểu là kiểu mình khai báo generic với Future.

- catchError tương tự với then nhưng sẽ bắt lỗi khi Future kết thúc.

- whenComplete đọc qua tên cũng hiểu rồi, nó sẽ được chạy Future kết thúc.

* 1. **Async-await:**

- async đánh dấu function sẽ thực hiện bất đồng bộ.

- await là sẽ chờ kết quả thực hiện của một async task trả về

- Các câu lệnh khác ở dưới await trong async function thì sẽ dùng chờ await thực hiện xong. Thì mới tới lượt mình thực thi.

* 1. **Handling Error:**

- xử lý lỗi nếu có.

- try cố gắng thực thi việc gì đó

- catch sẽ bắt lỗi được phát sinh ra

- finally dù lỗi hay không thì sẽ vẫn thực thi cuối cùng

- cần phối hợp try/catch với async/await.

* 1. **API:**

Các bước để lấy data từ 1 API Restful:

- Create model: để save data từ api.

- Create request: để lấy response từ một link API.

- Parse data: phân tích dữ liệu JSON nhận được thành các đối tượng mà chúng ta sẽ dùng.

- Handling error:

* 1. **Stream:**

- tạo ra một dòng chảy dữ liệu. Các giá trị sẽ đến từ tương lai.

- sử dụng rộng rãi trong Asynchronous.

- Dùng trong các trường hơp: reading large files, download file from server, listening events, showing change from state to UI...

- có thể tạo các Stream riêng hoặc sử dụng những Stream mà Dart cung cấp.

- Subcribing to a stream: using a lib dart:io

+ Reading as a string:

+ Reading from a stream: when the content of file is large, it needs to using a Stream to transfer data with each package. Then we will handle the packages.

- Transforming a stream: chuyển đổi data from stream to basic data types.

- Error Handling: 2 solutions: using callback or tryCatch.

- Canceling a stream:

- Creating a custom stream:

1. **Notes:**

- Sealed Classes:

+Các lớp "niêm phong" hay "kín" ( Sealed ) cho phép một lớp hoặc giao diện kiểm soát những lớp nào có thể triển khai hoặc mở rộng nó.

+ Một class không được class Account cho phép sẽ gặp phải lỗi biên dịch nếu nó cố gắng mở rộng nó.

- Final class khác gì với sealed class:

+ Final class: có thể được coi là một hình thức "niêm phong" khác hạn chế tất cả các lớp được mở rộng lớp mục tiêu

+ final: Không thể mở rộng thêm.

+ sealed: Chỉ có thể được mở rộng bởi các lớp con được phép (permitted subclasses) của nó. Bằng cách đó, chúng ta có thể hạn chế thêm phân lớp (subclassing). các permitted classes phải ở cùng mô-đun với lớp "niêm phong" (nếu lớp "niêm phong" nằm trong mô-đun đã đặt tên) hoặc trong cùng một package.