Chương 2

Tổng quan về kiến trúc của di động

Mục lục

Mục lục

1. Kiến trúc hệ thống

- Khi nhắc đến kiến trúc cho một ứng dụng di động, ta phải quan tâm đến cả kiến trúc của phần cứng thiết bị và kiến trúc của cả hệ thống server hỗ trợ.
- Cả **phần cứng + ứng dụng di động + server** phải tạo thành một hệ thống đồng nhất, trong suốt và khả chuyển
- Tính đồng nhất: các dữ liệu dễ dàng trao đổi qua các thành phần khác nhau
- Tính trong suốt: các thành phần dễ dàng phát hiện được hoặc hiểu được các lỗi của thành phần khác trả về.
- Tính khả chuyển: dễ dàng thay đổi thành phần này bằng một thành phần khác mà không mất nhiều thời gian chỉnh sửa.

1.1. Lịch sử ngành lập trình di động

- Thiết bị di động ngày càng có các thay đổi về thiết kế:
  - Điện thoại dạng bag.
  - Tiếp theo là thế hệ có thể cầm gọn trong lòng bàn tay
  - Điện thoại vỏ gập
  - Đầy đủ bàn phím
  - Điện thoại thông minh với màn hình cảm ứng và rất ít nút nhấn
- Các thiết bị ngày nay có màn hình ngày càng to, thời lượng lớn, thậm chí cồng kềnh đến mức không áp vào tai mà nghe gọi được như điện thoại (iPad)

1.1. Lịch sử ngành lập trình di động (2)

- Motorola DynaTAC 8000X
  - 1983
  - 13 x 1.75 x 3.5
  - 2.5 pounds
  - $3,995
  - + Monthly Fee
  - + Pay per minute

1.1. Lịch sử ngành lập trình di động (3)

- Từ những cục gạch thô kệch, điều gì đã khiến chúng ta giờ đây phải quan tâm đến trải nghiệm của người dùng trên di động:
  - Giá thành rẻ đi khiến nó không còn là thứ xa xỉ.
  - Pin thời lượng lâu hơn, kích thước nhỏ gọn đi, công nghệ cảm ứng nhạy hơn, màn hình chống vỡ, chống xước tốt hơn...
  - Các nhà sản xuất phần cứng di động không muốn và cũng không còn khả năng viết ra các ứng dụng đáp ứng cho người dùng nữa.
  - Tuy vậy các nhà sản xuất phần cứng không muốn chia sẻ quá nhiều về bí mật công nghệ để sản xuất phần cứng của mình
  - Từ đó xuất hiện các chuẩn được thống nhất giữa các nhà sản xuất phần cứng và nhà phát triển ứng dụng.
  - Đầu tiên là kể đến các chuẩn Web trên di động.

1.1. Lịch sử ngành lập trình di động (4)

- WAP (Wireless Application Protocol)
  - Là một chuẩn con của HTTP để trao đổi dữ liệu giữa di động (kết nối kém, không ổn định) và server
  - Sử dụng WML thay vì HTML
  - Hai trang sử dụng WAP phổ biến nhất: CNN và ESPN

1.1. Lịch sử ngành lập trình di động (5)

- Thanh toán trên thiết bị di động thời điểm đó (và vẫn còn được sử dụng thời nay) là SMS, với các tin nhắn được gọi là giá trị gia tăng (VAT)
  - Gửi tin đến một đầu số nào đó để được phục vụ với phí tin nhắn cao vọt lên hẳn so với tin bình thường
- Có những đơn vị chấp nhận thanh toán qua thẻ cào hoặc Web charging.
- Dần dần theo thời gian, nhu cầu thanh toán qua điện thoại càng tăng cũng như các công ty lớn nhìn thấy được tiềm năng của thị trường điện thoại (trừ Microsoft)
- Sự phát triển của phần cứng khiến ngôn ngữ lập trình cho di động không còn là ác mộng như J2ME nữa.

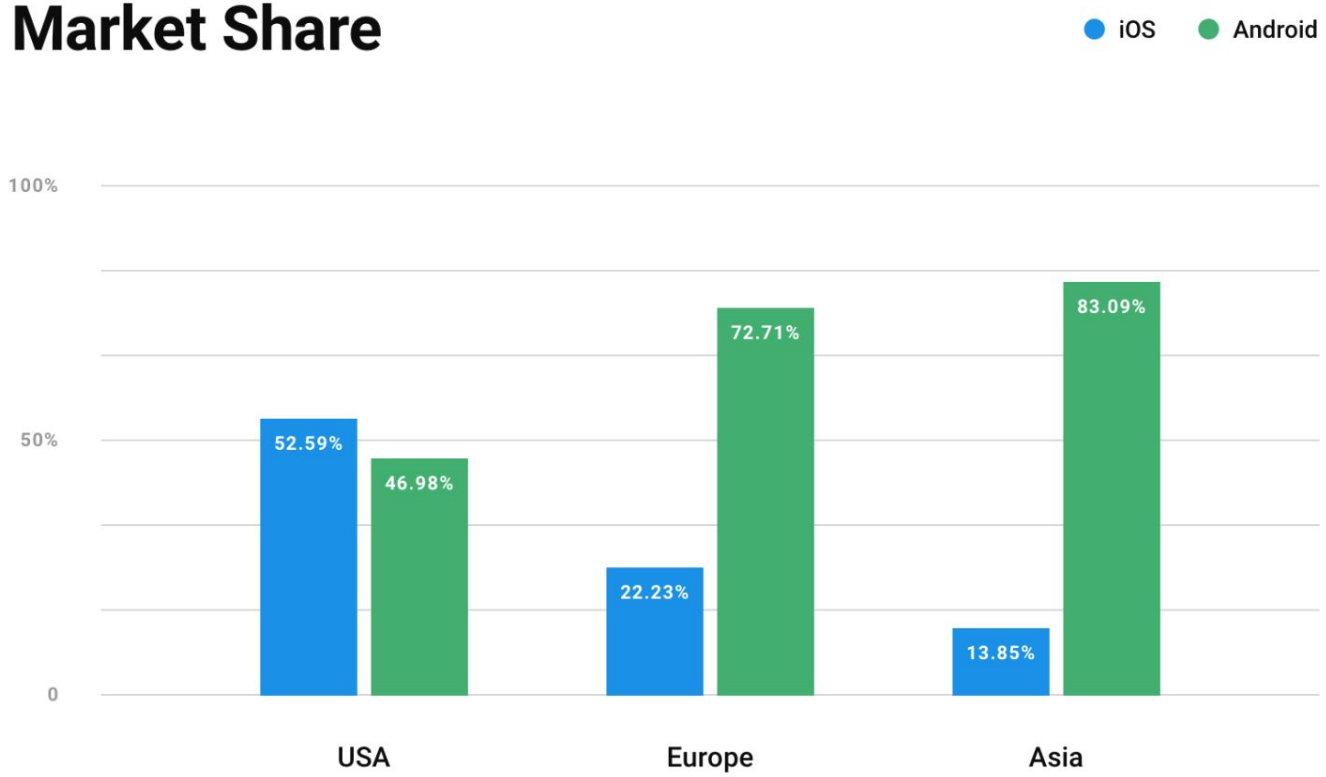
1.1. Lịch sử ngành lập trình di động (6)

- Trước năm 2010, có thể nói Nokia chiếm ngôi vương trên thị phần điện thoại toàn cầu nói chung và Việt Nam nói riêng
- Các nhà sản xuất khác chia đều thị phần còn lại như Blackberry, Samsung, HTC…
- Mỗi nhà sản xuất đều cố thử hệ điều hành của riêng mình, và họ dường như không dốc toàn lực phát triển chúng.
- Nhiều hệ điều hành ra đời, chết yểu nhưng các nhà phát triển vẫn rất đủng đỉnh với công việc của mình (Windows CE, Windows Mobile, Linux, Symbian…)
- Mọi việc thay đổi hoàn toàn khi iPhone ra đời.

1.1. Lịch sử ngành lập trình di động (7)

- Từng có trên thị trường ba ông lớn về sản xuất các ứng dụng di động và cả thiết bị di động:
- Microsoft:
  - Với bộ phận di động của Nokia, cố gắng với triết lý Windows Universal.
  - Do đã chậm trễ nên sản phẩm Windows Phone cuối cùng cũng thất bại
- Apple:
  - Vẫn luôn là đỉnh cao trong thiết kế phần cứng, phát triển ứng dụng và kinh doanh ở mảng di động
- Google:
  - Vẫn tuân theo tiêu chí chuẩn mở của Android đời đầu
  - Là một trong tập đoàn lớn nhất thế giới, vẫn thành công trong mảng điện thoại này

Market Share

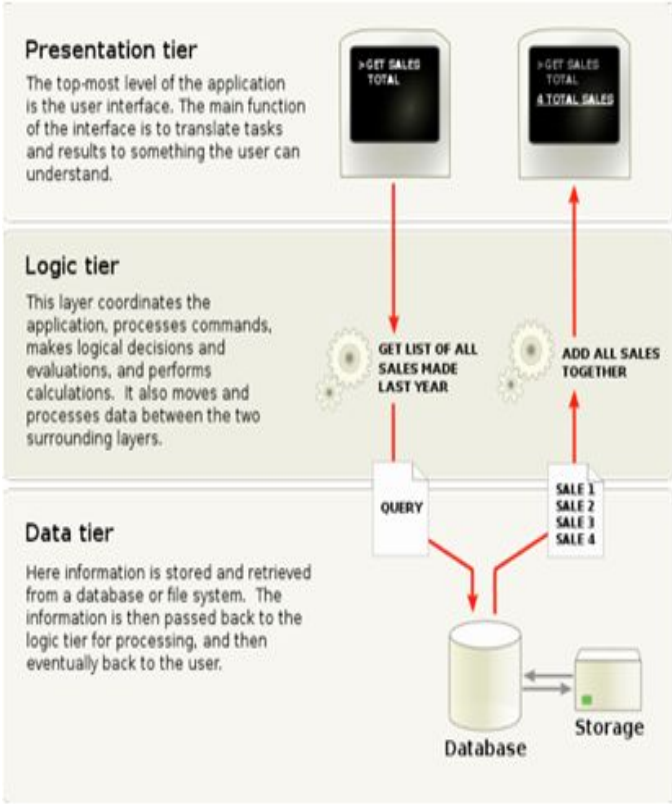


1.1. Lịch sử ngành lập trình di động (8)

- Hiện tại còn lại hai hệ điều hành di động phổ biến:
  - iOS:
    - Hoạt động trên thiết bị iPhone và iPad
    - Dùng ngôn ngữ Objective-C hoặc Swift với XCode hoặc các framework đa nền tảng (Unity, Xamarin, Flutter…)
    - Vẫn luôn được Apple quản lý chặt chẽ với triết lý "đóng"
  - Android:
    - Rất đa dạng về mẫu mã và chủng loại của cả phần cứng và hệ điều hành
    - Lập trình bằng Java dùng Android Studio hoặc các bên đa nền tảng khác (Unity, Xamarin, Flutter...)
    - Mã nguồn mở, các nhà sản xuất khác có thể tùy chỉnh được (như Samsung, HTC, Sony...)

1.2. Kiến trúc phần mềm

- Cả hai hệ điều hành, đều cổ vũ việc sử dụng chung một số mẫu kiến trúc
  - Layers: ba thành phần khác nhau
  - LTV iOS ưa chuộng MVC (Model/View/Controller)
  - LTV Android ưa chuộng MVVM
  - Hỗ trợ trao đổi dữ liệu theo kiểu Client-Server

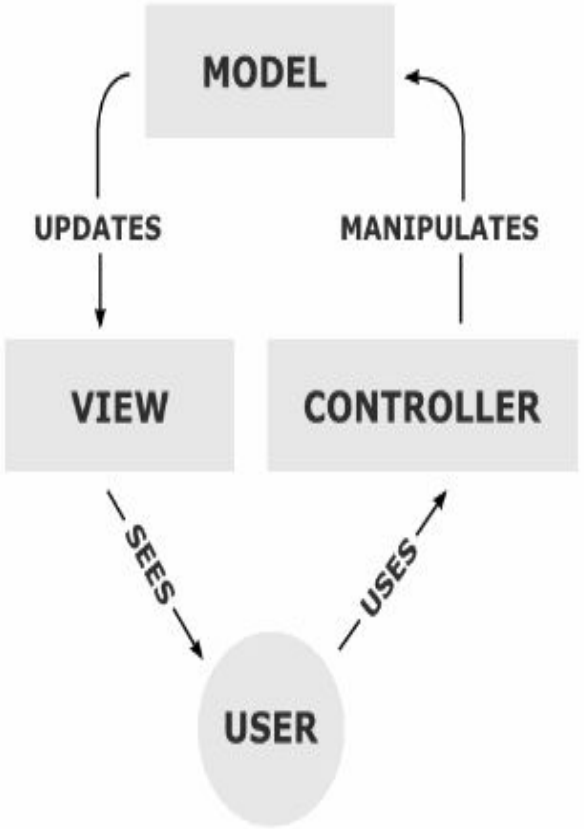


1.2. Kiến trúc phần mềm (2)

- Kiến trúc ba tầng được áp dụng cho nhiều kịch bản khác nhau?
  - Tầng trình diễn chỉ đảm nhiệm việc hiển thị dữ liệu, tăng trải nghiệm cho người dùng và dễ dàng lắp ghép với các phiên bản khác nhau của các tầng khác.
  - Tầng nghiệp vụ thực hiện việc chuẩn bị dữ liệu đầu vào để gửi các yêu cầu truy vấn dữ liệu, chỉnh sửa dữ liệu được nhận về và xử lý các lỗi trả về
  - Tầng dữ liệu: lưu trữ các dữ liệu quan trọng nhất, tổ chức truy vấn sao cho ra được kết quả nhanh nhất và chính xác nhất.

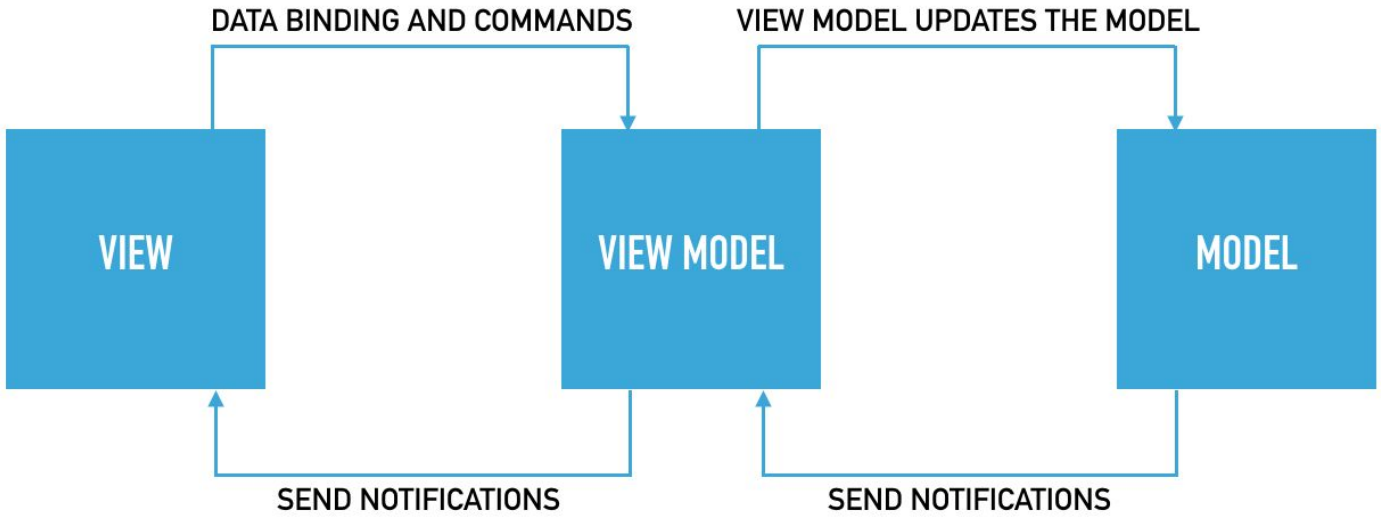
1.2. Kiến trúc phần mềm (3)

- Model/View/Controller
  - Được minh họa như hình bên đây
  - Mẫu MVC này phân công ra:
    - Tầng Model định danh ra những gì cần trả về cho người dùng
    - Tầng Controller đưa ra các yêu cầu truy vấn tài nguyên
    - Tầng View hiển thị lại các dữ liệu theo một chuẩn dễ hiểu cho người dùng.



1.2. Kiến trúc phần mềm (4)

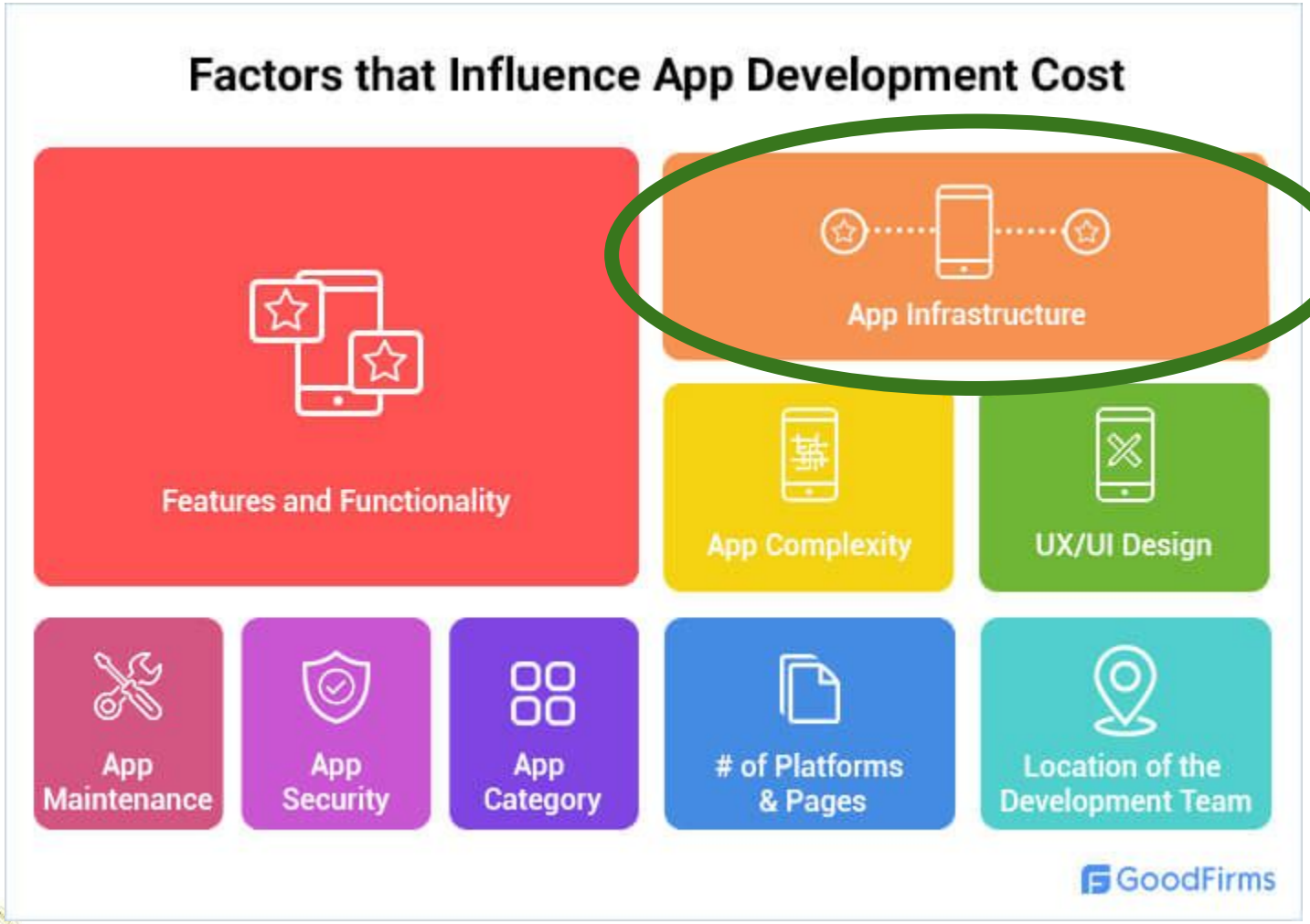
- Model/View/ViewModel
  - khá giống với MVC
  - ViewModel: Chứa các model và chuẩn bị các dữ liệu quan sát cho View. Nó cung cấp các cơ chế để truyền dữ liệu từ View sang Model.
  - Một cập nhật bên đối tượng của một lớp thuộc nhóm View sẽ cập nhật dữ liệu bên đối tượng của lớp thuộc nhóm Model và ngược lại (Data Binding)



1.2. Kiến trúc phần mềm (5)

- Kiến trúc client/server
  - Kiến trúc MVC/MVVM có thể áp dụng cho các ứng dụng không có kết nối Internet
  - Còn loại kiến trúc client/server đề cập đến các ứng dụng có truy cập đến máy tính ở bên ngoài, qua HTTP request hoặc socket hoặc webservice
  - Nhìn chung, các ứng dụng muốn được sử dụng lâu dài bởi người dùng, không thể không tuân theo kiến trúc client/server.
  - Tuy nhiên, do kết nối Internet vẫn có những lúc không ổn định hoặc gây tốn tài nguyên (bộ nhớ, dung lượng, tốc độ xử lý…) nên cần có lưu trữ dữ liệu cục bộ tại thiết bị.

1.2. Kiến trúc phần mềm (6)



Hourly rates of specialists according to Accelerance

Title of Employee	United States	Latin America	Eastern Europe	Asia
Business Analyst	$110 - $205	$45 - $55	$40 - $63	$30 - $42
Architect	$198 - $292	$60 - $72	$50 - $77	$35 - $48
Project Manager	$133 - $233	$55 - $66	$45 - $70	$35 - $48
Jr. Developer	$105 - $111	$35 - $44	$25 - $42	$18 - $24
Mid-Level Developer	$132 - $140	$30 - $52	$35 - $56	$24 - $35
Sr. Developer	$154 - $163	$45 - $55	$45 - $70	$30 - $42
Lead Developer	$176 - $187	$50 - $61	$45 - $70	$30 - $42
Junior QA	$77 - $81	$30 - $39	$25 - $42	$15 - $24
Mid-Level QA	$99 - $105	$35 - $44	$30 - $49	$20 - $30
Senior QA	$143 - $169	$40 - $50	$40 - $63	$25 - $36
Graphic Designer	$79 - $163	$40 - $50	$35 - $56	$25 - $36

Mục lục

2. Kiến trúc ứng dụng Android

- Tất cả ứng dụng Android được viết bằng ngôn ngữ Java (hoặc một ngôn ngữ khác nếu sử dụng công nghệ đa nền tảng)
- File chạy của ứng dụng Android có đuôi APK
- Khi được đăng tải trên Store, Google sẽ cập nhật một chữ ký vào file chạy này.
- Ngoài ra bạn có thể sử dụng dịch vụ App signing để lưu trữ các khóa bí mật (private key) của mình.

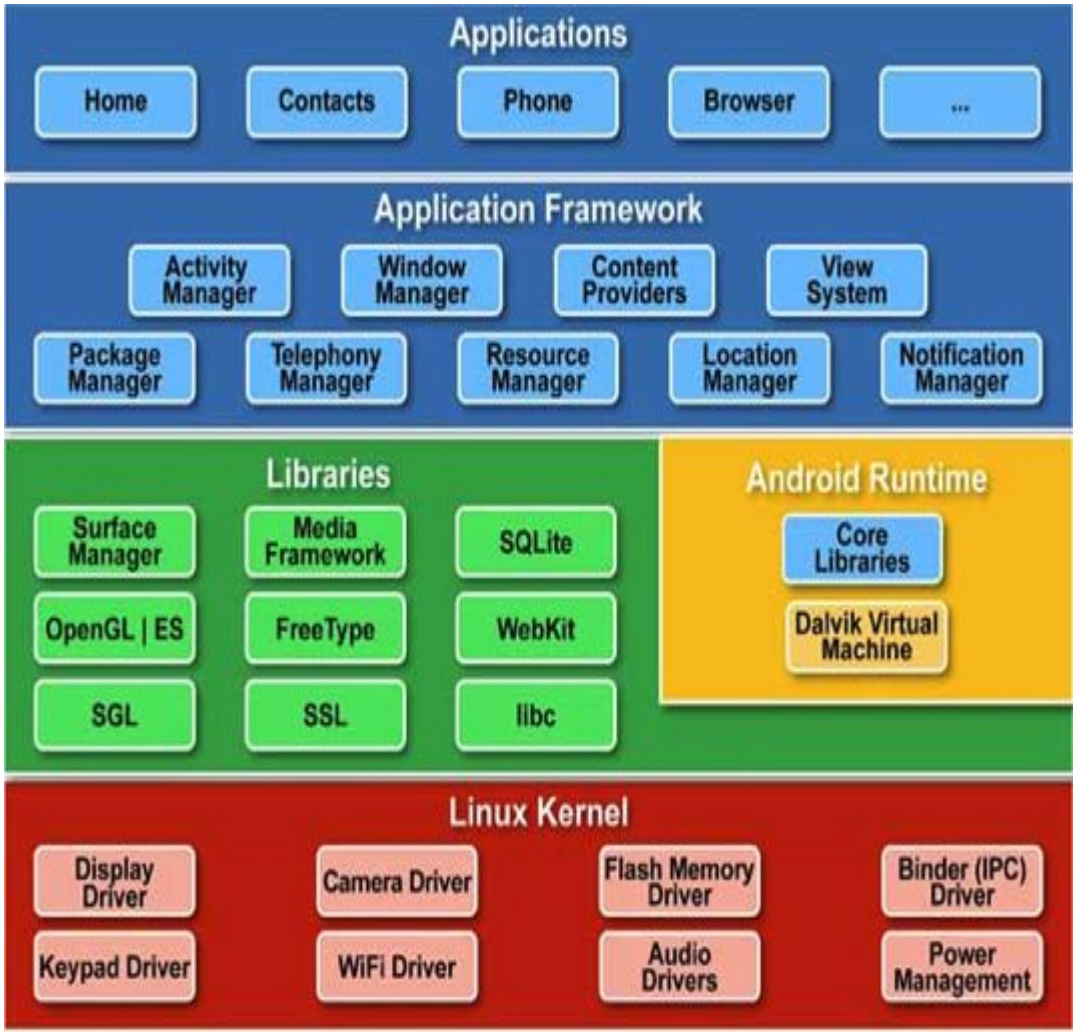
2. Kiến trúc ứng dụng Android (2)

- Android là một hệ điều hành đa người dùng, phát triển trên nền tảng Linux
- Mỗi ứng dụng được gắn liền với một định danh người dùng khi nó được cài
- Mỗi ứng dụng có một khu vực lưu trữ của riêng nó "sandbox" nơi mà các file được gán quyền chỉ dành cho ứng dụng đó thực hiện việc đọc ghi
- Mỗi ứng dụng chạy trên một máy ảo VM của riêng nó (tức là mỗi mã nguồn của một ứng dụng chạy độc lập với các ứng dụng khác)
- Mỗi ứng dụng có một mã tiến trình riêng.

2. Kiến trúc ứng dụng Android (3)

- Mỗi ứng dụng Android vẫn có thể chia sẻ dữ liệu cho các ứng dụng khác
- Các ứng dụng khác nhau vẫn có thể chung user_id nếu chúng có chung chứng chỉ số (digital certificate).
- Mỗi ứng dụng có thể thiết lập việc chia sẻ dữ liệu cho các ứng dụng khác dựa trên chỉnh sửa XML permission trong file Manifest.
- Mỗi ứng dụng vẫn có thể có quyền truy cập được địa chỉ, trạng thái điện thoại, SMS… khi người dùng đồng ý cấp quyền.

2. Kiến trúc ứng dụng Android (4)



2. Kiến trúc ứng dụng Android

CÂU HỎI: Việc Android lại sử dụng Java để lập trình có ưu điểm gì?

- Hầu như các kĩ sư CNTT đều biết lập trình Java
- Java vẫn là một ngôn ngữ, trong sáng, cú pháp chặt chẽ, strong type
- Có máy ảo hỗ trợ để đảm bảo việc "**write once, run everywhere**"
- Có rất nhiều công cụ và diễn đàn hỗ trợ
- Không cần phải quan tâm đến việc giải phóng bộ nhớ
- Mục đích ban đầu của Java là hoạt động trên các thiết bị gia dụng và cầm tay

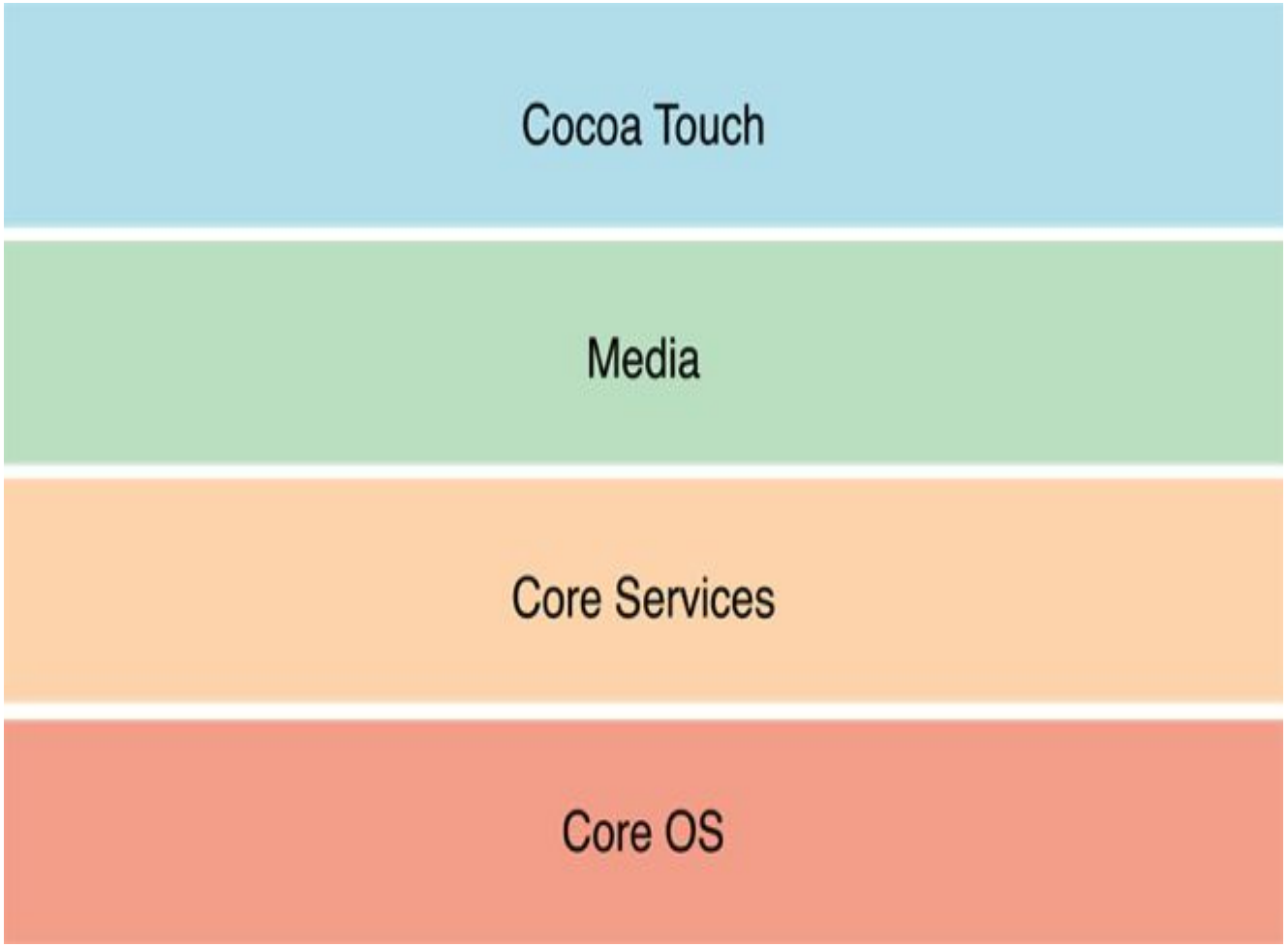
Mục lục

1. Kiến trúc hệ thống
2. Kiến trúc ứng dụng Android
3. **Kiến trúc ứng dụng iOS**
4. Kiến trúc đa nền tảng

3. Kiến trúc ứng dụng iOS

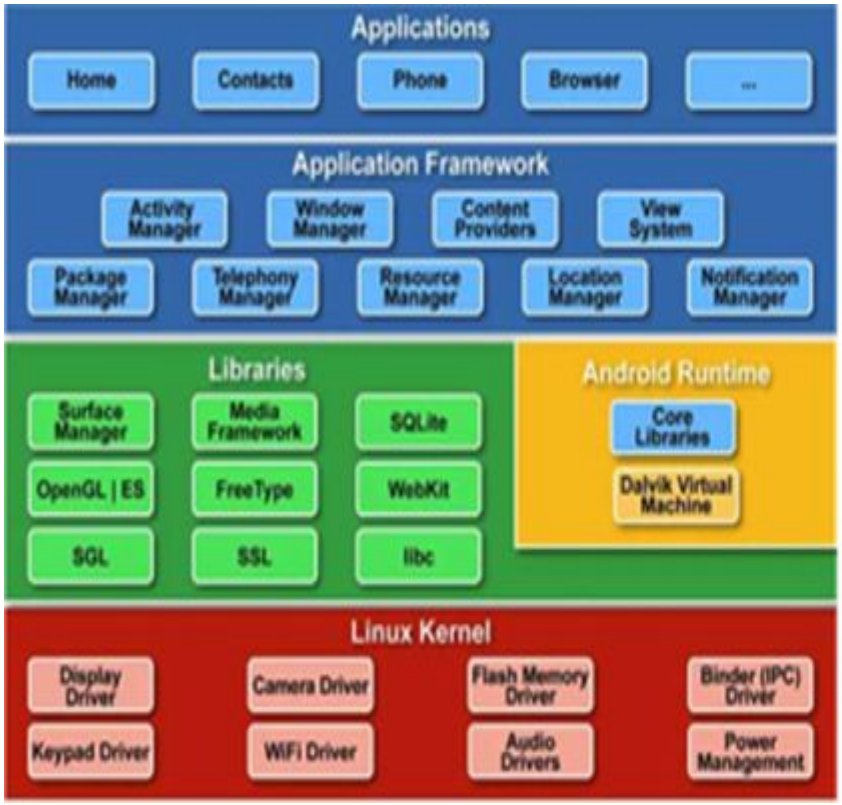
- Các đặc tính nổi bật nhất của iOS:
  - Cũng thuộc nhóm hệ điều hành *nix
  - Cùng thuộc nhánh phát triển như OS X của Apple
  - Dựa trên nền tảng Cocoa, phát triển thành Cocoa Touch - chứa các API chính để ứng dụng tương tác với hệ điều hành
  - iOS cũng phân chia thành các tầng như Android, nhưng nhìn chung nó đơn giản hơn nhiều.

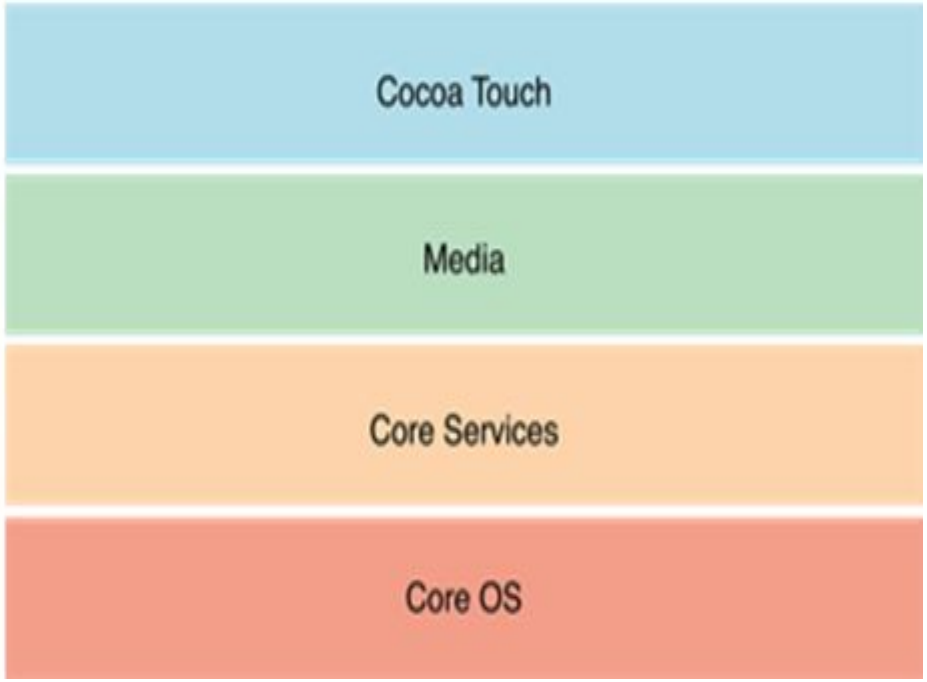
3. Kiến trúc ứng dụng iOS



3. Kiến trúc ứng dụng iOS

Hãy nhìn vào và so sánh?



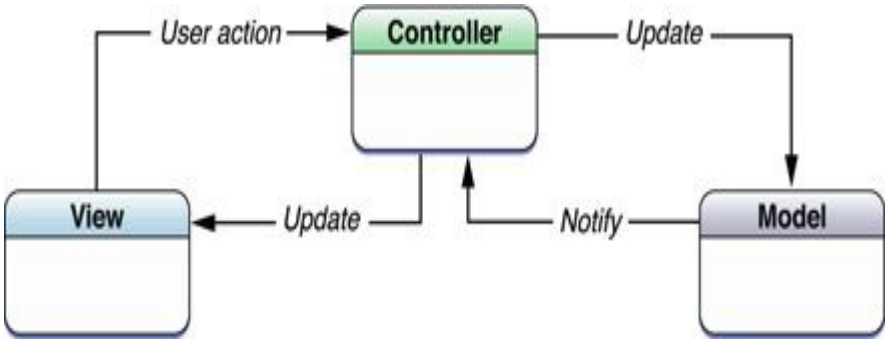


3. Kiến trúc ứng dụng iOS

- Sự khác biệt giữa hai nhóm hệ điều hành này là:
  - Mỗi tầng của iOS chỉ có một thành phần tham gia
  - Tầng Media của iOS liên quan đến dữ liệu đa phương tiện được sắp xếp thành một tầng riêng
  - iOS không cần tầng liên quan đến máy ảo Java
  - Tuy vậy, có khả năng kiến trúc của iOS chưa được công bố rõ ràng như Android.

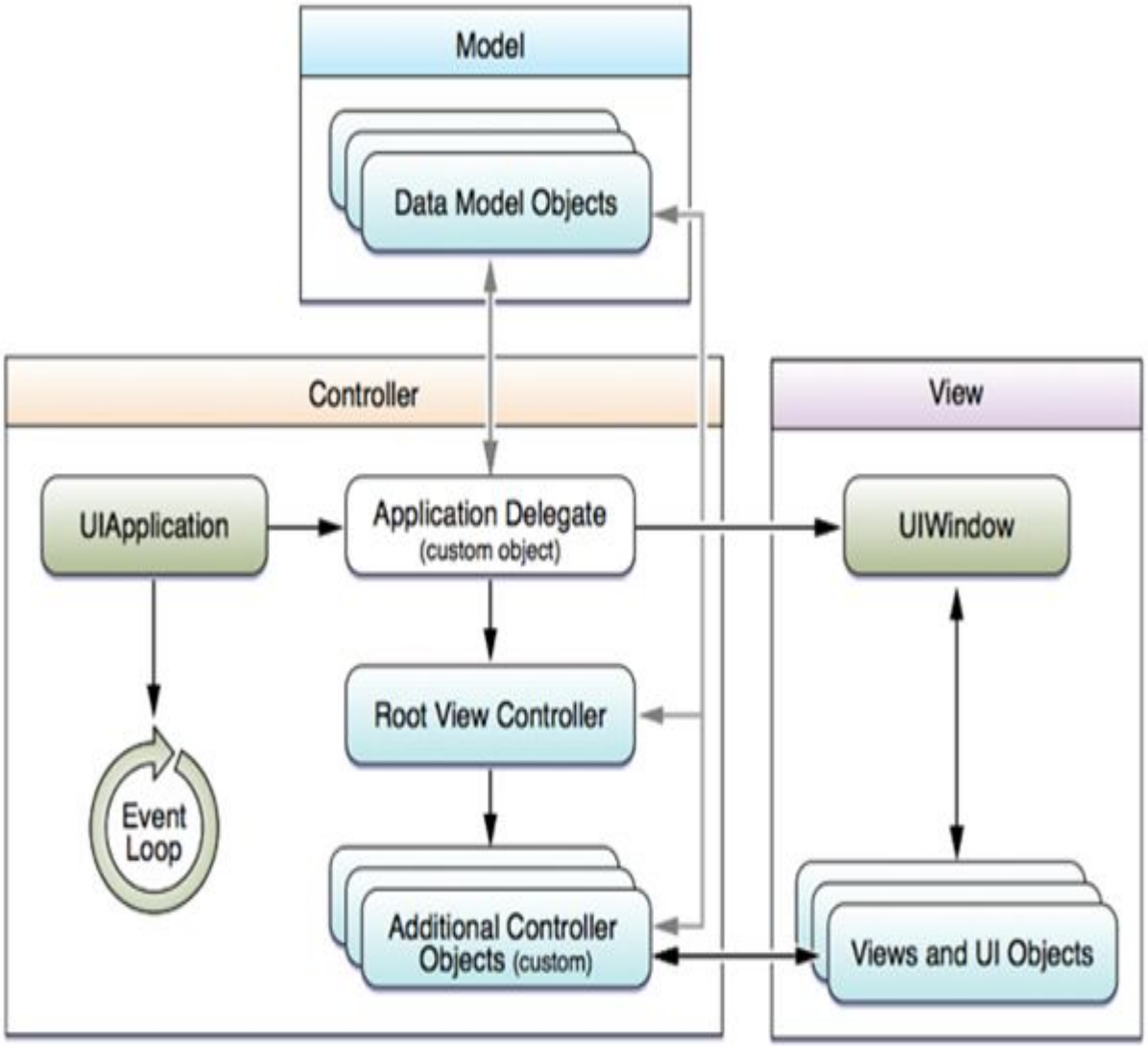
3. Kiến trúc ứng dụng iOS

- MVC trong iOS:
  - Vẫn Model-View-Controller là một phương pháp thiết kế ứng dụng cơ bản cho thiết bị di động
  - Do khả năng hướng đối tượng của Objective-C, MVC là một trong những mẫu thiết kế cho cả iOS và OS X



3. Kiến trúc ứng dụng iOS

MVC trong iOS:



Mục lục

1. Kiến trúc hệ thống
2. Kiến trúc ứng dụng Android
3. Kiến trúc ứng dụng iOS
4. **Kiến trúc đa nền tảng**

4. Kiến trúc đa nền tảng

Nếu không sử dụng các công cụ đa nền tảng, cần phải phát triển riêng từng nền tảng

Công cụ	Android	iOS
IDE nguồn mở	Android Studio	N/A
IDE bản quyền	IntelliJ IDEA	XCode
Công cụ kiểm thử	JUnit, Espresso	XCUnit
Công cụ đo đạc hiệu năng	Android Profiler	XCode Instruments
Công cụ quản lý mã nguồn	Git, Subversion, CVS	
Bộ giả lập	Android Emulator, Intel HAXM, Genymotion	iPhone Simulator

4. Kiến trúc đa nền tảng

- Khái niệm: là ứng dụng được tạo ra bởi các framework hỗ trợ cross-platform "Viết một lần chạy trên nhiều thiết bị"
- Đánh giá: để quyết định phát triển ứng dụng đa nền tảng hay không? Đội phát triển cần tự trả lời một số câu hỏi:
  - Nhóm người dùng nào bạn định nhắm đến (nhóm Android hay iOS hay cả hai)
  - Bạn định đổ bao nhiêu tiền vào chi phí phát triển?
  - Người dùng mong đợi trải nghiệm cao cấp đến mức nào?
  - Bạn muốn sử dụng loại phương án thanh toán nào?

4. Kiến trúc đa nền tảng (2)

- Đội phát triển cần tự trả lời một số câu hỏi (tiếp):
  - Liệu có đang cố gắng giảm chi phí phát triển cho cả hai nền tảng không?
  - Hiện tại có một nhóm phát triển cho vài nền tảng hay mỗi nền tảng một nhóm?
  - Ai sẽ bảo trì hệ thống này?
- Câu trả lời (của đa số nhóm được khảo sát) là:
  - Muốn phục vụ càng nhiều người dùng càng tốt
  - Muốn chi phí càng rẻ càng tốt
  - Muốn bảo trì càng dễ càng tốt.
- Với ba tiêu chí: rẻ, nhanh, tốt thì chỉ chọn được 2.
- Ứng dụng đa nền tảng nhìn chung là rẻ, nhanh nhưng chưa thực sự tốt

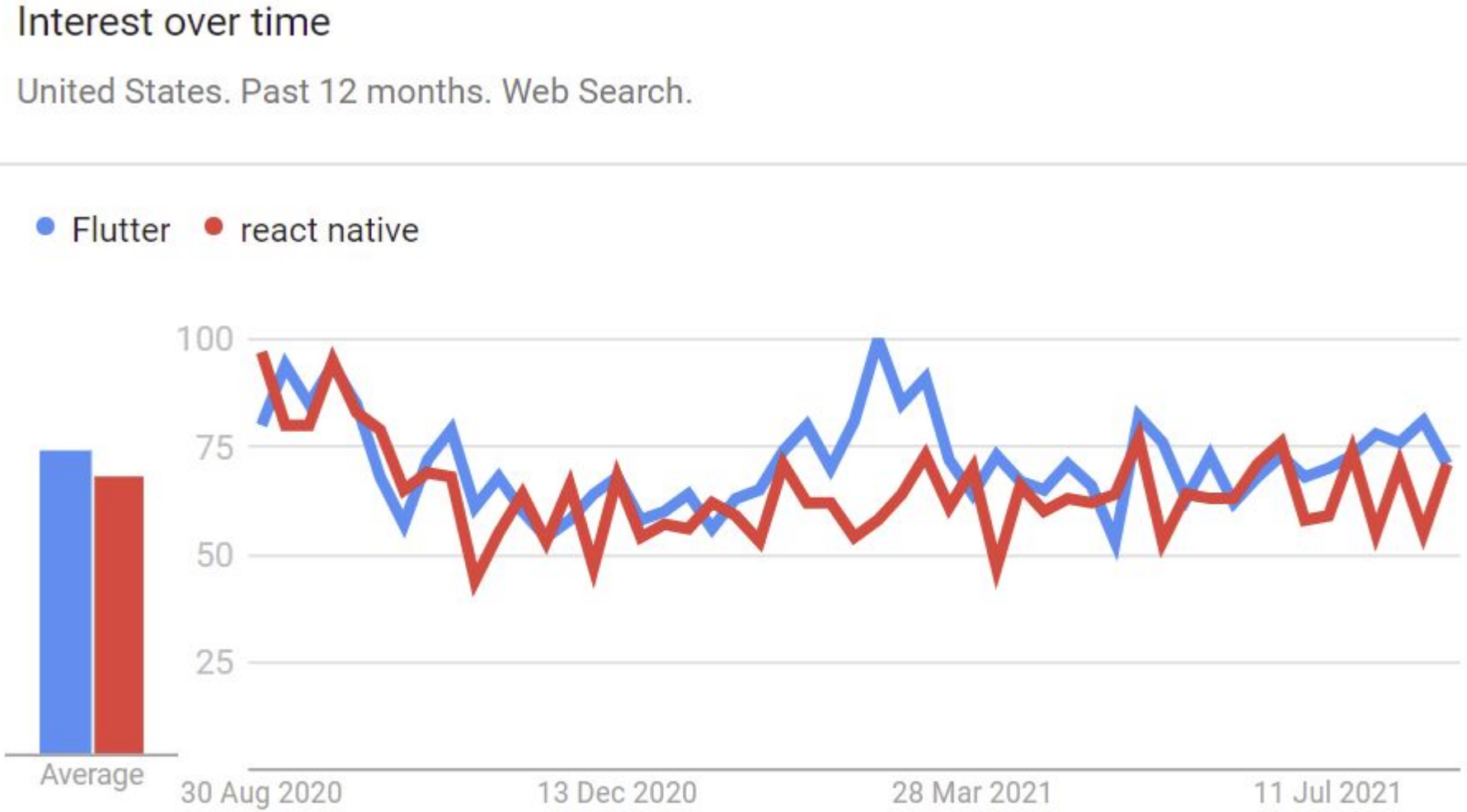
4. Kiến trúc đa nền tảng (3)

- Những chương trình đa nền tảng trước đây đều không tốt:
  - Giao diện người dùng nghèo nàn
  - Nghèo về hiệu năng
  - File chạy có kích thước lớn
  - Các nhà phát triển nền tảng đều không quan tâm đến chúng

4. Kiến trúc đa nền tảng (4)

- Những chương trình đa nền tảng hiện nay đều đã có các cải thiện đáng kể:
  - Lập trình viên có thể tạo ra các giao diện tùy biến tốt
  - Nhiều công cụ đa nền tảng với các ông lớn công nghệ đứng đằng sau (Microsoft, Google, Facebook)
  - Kích thước file cài đặt đã được giảm xuống (với các kỹ thuật giảm dung lượng được hỗ trợ)
  - Nhiều công cụ khác hỗ trợ cho phát triển ứng dụng được xây dựng

4. Kiến trúc đa nền tảng (5)



4. Kiến trúc đa nền tảng (6)

- Cần cân nhắc kĩ lưỡng với việc phát triển ứng dụng đa nền tảng:
  - Nếu được lên kế hoạch, từ 50-80% mã nguồn có thể dùng lại cho dự án khác. Khiến việc phát triển ứng dụng nhanh hơn và chi phí rẻ hơn.
  - Bảo trì có thể dễ dàng hơn nếu chỉ có vấn đề trong cài đặt mô hình nghiệp vụ, không phải ở giao diện
  - Kiểm thử mô hình nghiệp vụ có thể dễ dàng hơn hoặc tập trung hơn.

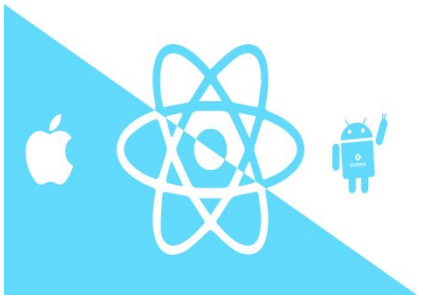
- https://www.infoq.com/articles/mobile-cross-platform-app-development

4. Kiến trúc đa nền tảng (7)

- Cần cân nhắc kĩ lưỡng với việc phát triển ứng dụng đa nền tảng:
  - Nếu đội phát triển đã nắm được Dart hoặc các ngôn ngữ như Javascript hoặc HTML5 thì không phải mất chi phí đào tạo
  - Thiết kế giao diện, có thể làm ra những sản phẩm không tuân theo quy tắc ngầm định của nền tảng, khiến người dùng ban đầu bỡ ngỡ.
  - Nền tảng có thể có các thay đổi và các công cụ không theo kịp
  - Hiệu năng vẫn là một vấn đề
  - Chi phí ban đầu và lợi nhuận thu được vẫn là câu hỏi lớn

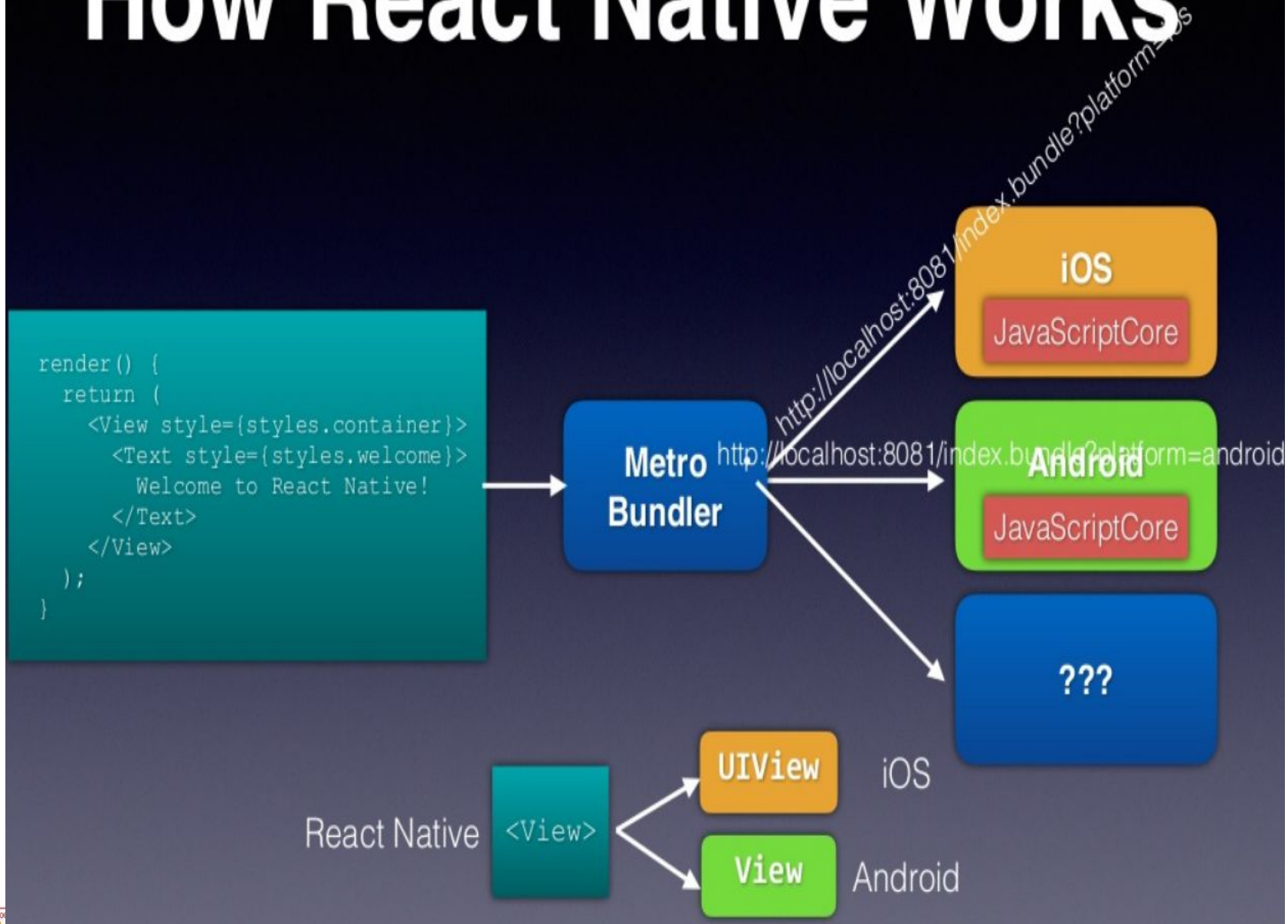
4. Kiến trúc đa nền tảng (8)

- React Native là một framework cho phép viết bằng Javascript để xây dựng ứng dụng cho iOS và Android.
  - Dựa trên thư viện React của Facebook để xây dựng giao diện ứng dụng
- Ứng dụng React Native được tạo dựng bởi:
  - Javascript
  - CSS
  - JSL = Javascript + XML



- Các mã nguồn javascript sẽ được chuyển đổi sang mã native của nền tảng.

How React Native Works



- UI rendered based on the platform's UI widget

4. Kiến trúc đa nền tảng (9)

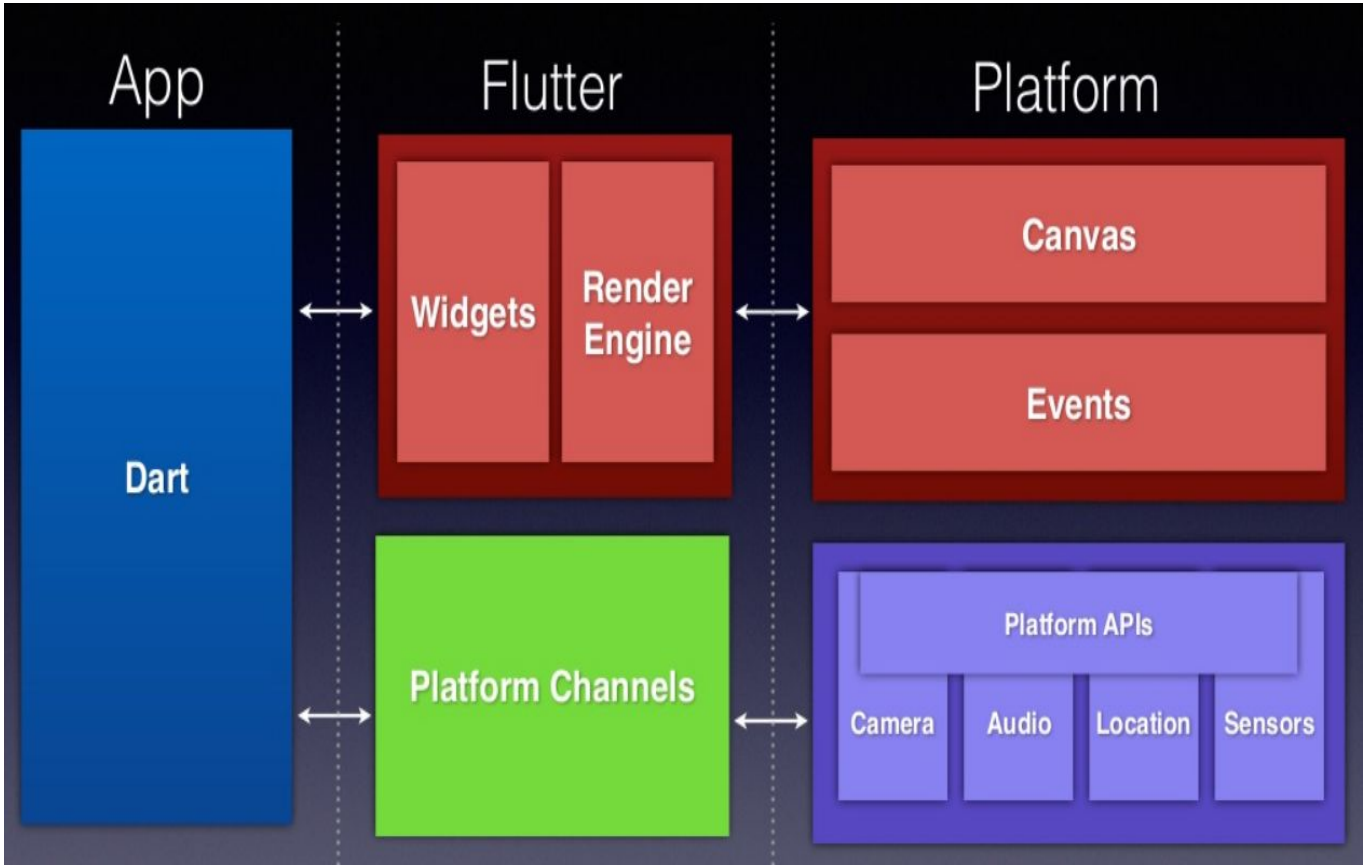
- **Flutter** là một bộ công cụ khả chuyển của Google để xây dựng các ứng dụng hoạt động trên mobile, web và máy tính cá nhân
- Dùng ngôn ngữ lập trình **Dart** của Google
- Flutter có các thành phần chính sau:
  - **Flutter engine**: viết bằng C++ cung cấp các hỗ trợ render mức thấp cho thư viện đồ họa Skia của Google
  - Thư viện nền tảng - viết bằng **Dart**, cung cấp các tầng cơ bản cài đặt chức năng cho ứng dụng và API để giao tiếp với Flutter engine
  - **Widgets** - các khối giao diện cơ bản cho UI

4. Kiến trúc đa nền tảng (10)

- Flutter không như Xamarin và React Native, nó không sử dụng các native Widgets của từng nền tảng.
  - Thay vào đó Flutter sử dụng các widget của riêng nó (gồm cả các thành phần của Material Design và Cupertino (iOS)), được quản lý và render lên màn hình bằng Flutter engine.
  - Về mặt này, Flutter sẽ tạo ra các ứng dụng ít phụ thuộc vào nền tảng hơn so với React Native.

4. Kiến trúc đa nền tảng (11)

Flutter không phụ thuộc vào widget của nhà sản xuất thiết bị gốc, nó tự hiển thị các giao diện đồ họa dựa trên engine hiệu năng cao của riêng nó.



4. Kiến trúc đa nền tảng (13)

- Dart là một ngôn ngữ lập trình được phát triển bởi Google
  - Tương tự như Java, C#, Swift, hay Kotlin
  - Để phát triển, Dart sử dụng trình biên dịch JIT (Just In Time)
  - Để xuất bản ứng dụng, Dart sử dụng trình biên dịch AOT (Ahead Of Time) để tạo ra mã chạy với hiệu năng cao nhất có thể
  - Dart cũng có thể biên dịch ra mã Javascript để hoạt động trên nền tảng Web

Scrolling

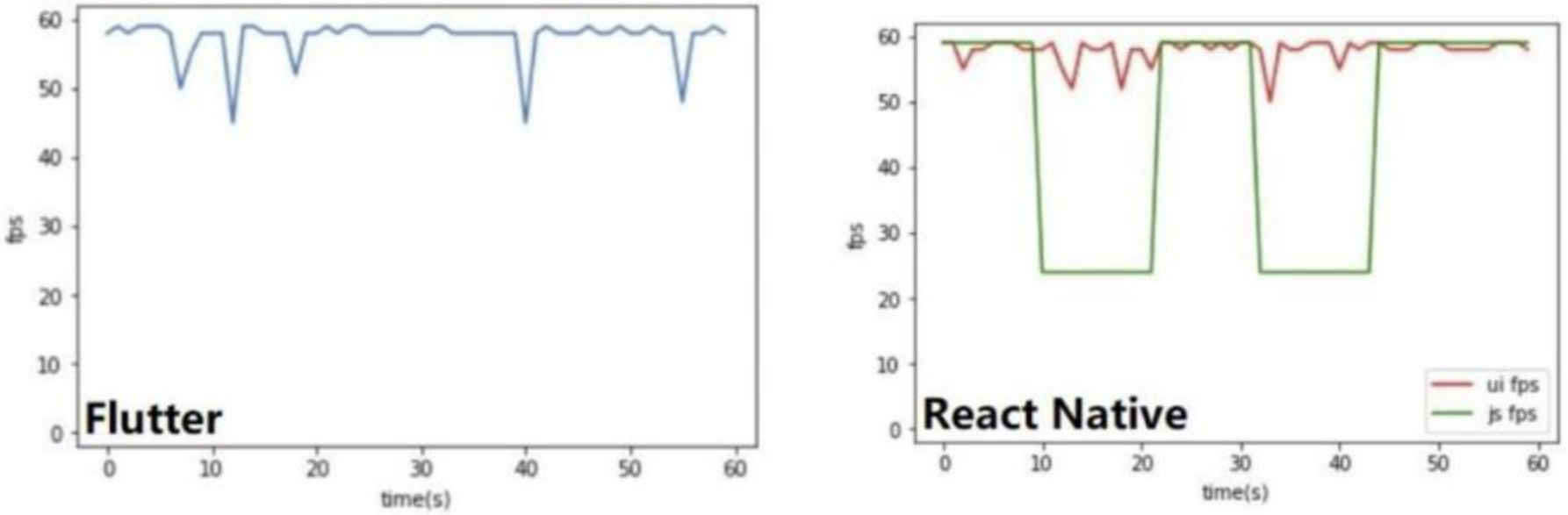
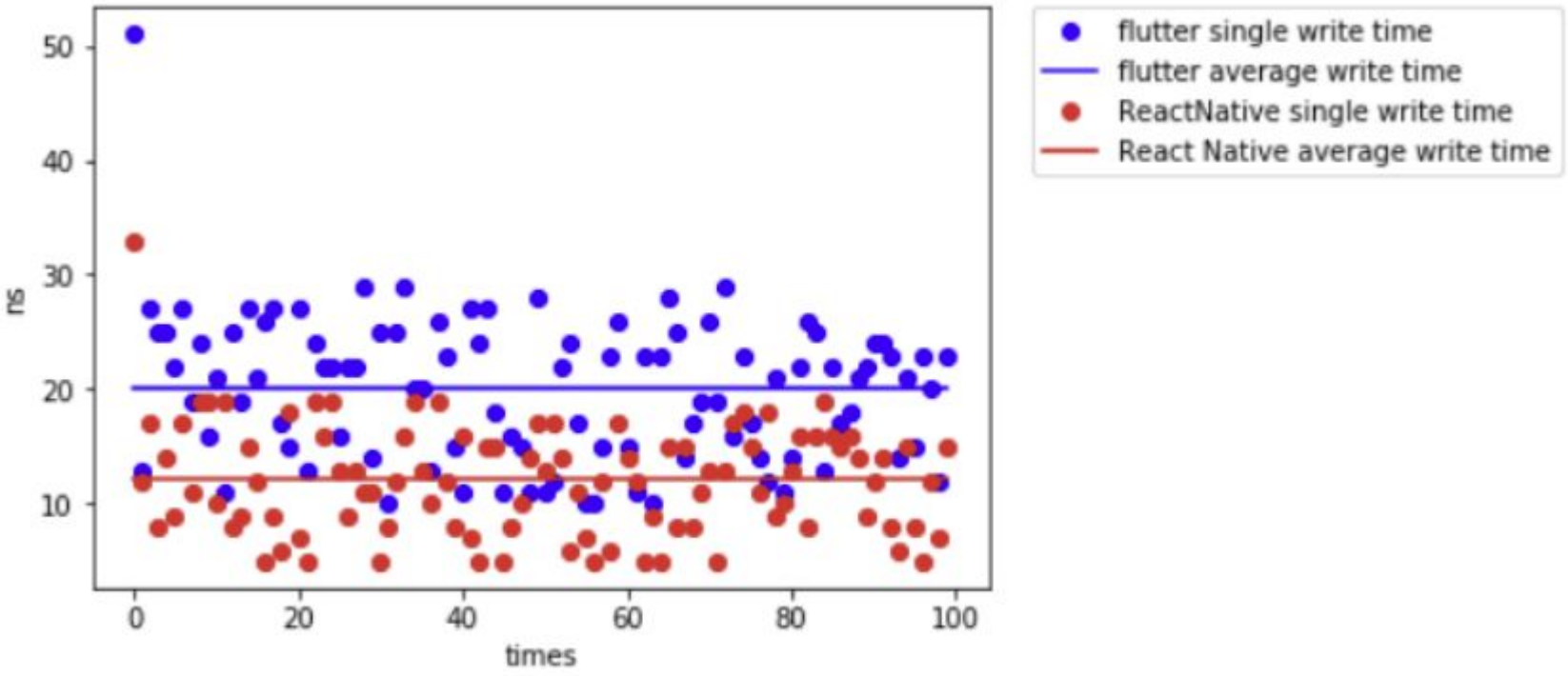


Figure 12 - FPS tracking for scrolling (Wenhao, 2018)

Wenhao, W. (2018, 03 01). React Native vs Flutter, cross-platform mobile application frameworks.

Ghi dữ liệu



Wenhao, W. (2018, 03 01). React Native vs Flutter, cross-platform mobile application frameworks.

Theo số liệu mới nhất

Table 4.25: Weighting of frameworks on bridge performance, ordered by sum ∑ (higher is better).

Framework	TTC	CPU	PreRAM	RAM	ComputedRAM	∑
Native	5	4	6	6	3	24
MAML/MD^2	4	5	5	5	4	23
NativeScript	6	6	3	3	2	20
React Native	2	1	4	4	5	16
Flutter	3	3	1	2	6	15
Ionic (Cordova)	1	2	2	1	1	7

Biorn-Hansen, Andreas. "Presence and Performance of Mobile Development Approaches." PhD diss., Brunel University London, 2021.

HẾT TUẦN 2

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict