



Chương 7

Đánh giá hiệu năng ứng dụng đa nền tảng

Mục tiêu

Sau bài học này sinh viên sẽ:

- + Có cái nhìn tổng quan và giải thích được các vấn đề hiệu năng của các công nghệ đa nền tảng
- + Nắm được nguyên lý chung để thực hiện việc đánh giá tài nguyên sử dụng trong một ứng dụng bất kỳ
- + Biết được các phương pháp luận tốt nhất để đo đặc hiệu năng ứng dụng

"Identifying Trade-offs Associated with Cross-platform Mobile Development Tools.", 2022

Mục lục

1. Giới thiệu
2. Các công cụ và phương pháp
3. Thời gian khởi chạy
4. Tài nguyên sử dụng
5. Trải nghiệm của đội phát triển
6. Kết luận

Tóm tắt nội dung

- ❖ Nghiên cứu này áp dụng cho 3 Cross Platform Development Technologies - CNDNT - hiện đại: Flutter, React Native, Progressive Web App – dựa trên mức độ quan tâm và chấp nhận trong những năm gần đây từ các cuộc khảo sát.
- ❖ Đánh giá định lượng để điều tra sự đánh đổi giữa CNDNT hiện tại liên quan tới: khả năng đáp ứng và sự tiện lợi khi sử dụng (responsiveness and convenience of use), sử dụng tài nguyên (resource use), hiệu suất (performance) và trải nghiệm của nhà phát triển (developer experience).
- ❖ Với mỗi phần thuộc khung đánh giá, phát triển một ứng dụng cụ thể cho từng OS (native Android, native iOS), sau đó triển khai trên 3 CNDNT hiện đại (Flutter, React Native, PWA), rồi so sánh chúng với nhau

*CNDNT: Công nghệ đa nền tảng

Mục lục

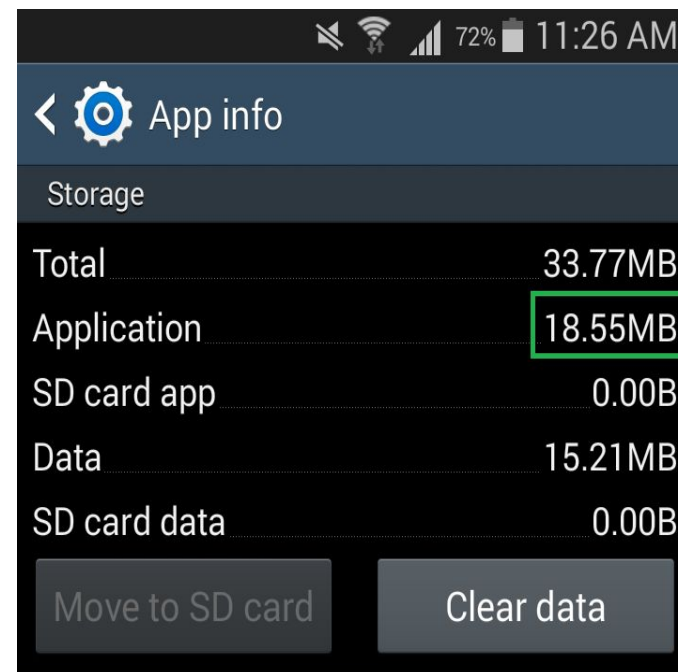
1. **Giới thiệu**
2. Các công cụ và phương pháp
3. Thời gian khởi chạy
4. Tài nguyên sử dụng
5. Trải nghiệm của đội phát triển
6. Kết luận

Giới thiệu các khái niệm

- Responsiveness : Khả năng đáp ứng.
 - + Ở đây có thể hiểu là khả năng mà ứng dụng hồi đáp cho người dùng khi ứng dụng được cài đặt bằng các công nghệ khác nhau



- Size of installer: Kích thước của ứng dụng cần tải xuống, ở đây được tính theo đơn vị Megabytes (MB).
 - + Nó sẽ phản ánh thời gian cài đặt, kích thước càng lớn thì thời gian cài đặt càng lâu
 - + Đây cũng là 1 yếu tố ảnh hưởng tới trải nghiệm lần đầu của người dùng



Storage	
Total	33.77MB
Application	18.55MB
SD card app	0.00B
Data	15.21MB
SD card data	0.00B
Move to SD card Clear data	

Giới thiệu các khái niệm sẽ xuất hiện

- **Launch time:** Thời gian để khởi chạy ứng dụng. Nó được tính từ thời điểm người dùng chạm vào biểu tượng ứng dụng đến thời điểm ứng dụng sẵn sàng phản hồi tương tác của người dùng
- **Navigation time:** Thời gian điều hướng của ứng dụng. Nó được hiểu là khoảng thời gian chuyển đổi giữa các trang. Nó được tính là thời gian người dùng bắt đầu điều hướng tới khi trang đích được tải và sẵn sàng tương tác



<https://github.com/Iman-Jamali/thesis-mobile-app-development-cross-platform-vs-native/tree/main/responsiveness-and-convenience>

Mục lục

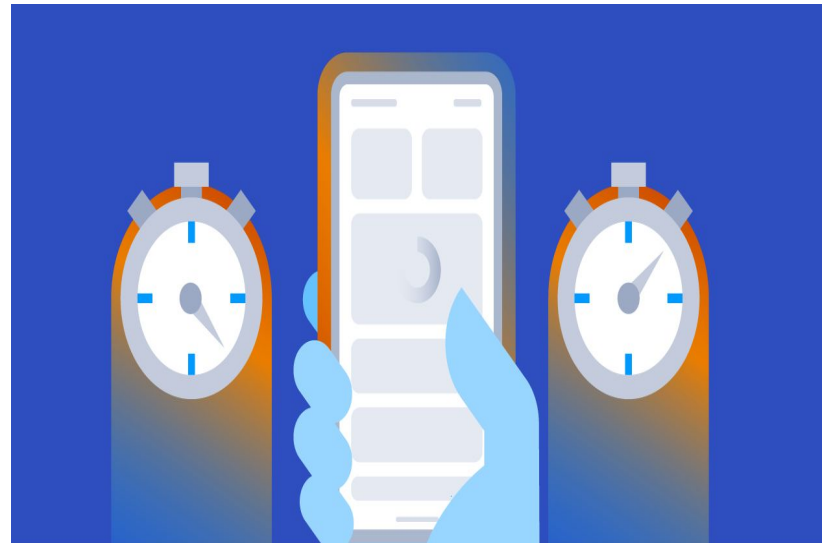
1. Giới thiệu
2. **Các công cụ và phương pháp**
3. Thời gian khởi chạy
4. Tài nguyên sử dụng
5. Trải nghiệm của đội phát triển
6. Kết luận

Các công cụ và phương pháp

Đo thời gian khởi chạy trên Android :

- ❖ Phương pháp đo được build cho React Native, Flutter, native Android
- ❖ 3 quy trình: cold start, warm start, hot start
- ❖ Trước tiên, loại bỏ tất cả các ứng dụng trong nền và khởi động lại thiết bị thử nghiệm một lần rồi bắt đầu đo bằng cách mở ứng dụng và ghi lại thời gian khởi chạy bằng Logcat

<https://blog.instabug.com/understanding-cold-hot-and-warm-app-launch-time/>



Các công cụ và phương pháp

Đo thời gian khởi chạy trên iOS :

- ❖ Phương pháp đo được build cho React Native, Flutter, native iOS
- ❖ 2 kiểu khởi chạy: cold launch, warm launch
- ❖ Sử dụng công cụ kiểm tra “Instruments” của Xcode để đo đạc. Sử dụng mẫu “App launch” để lập log và đo lường thời gian khởi chạy bằng cách kiểm tra log về thời gian mà app cần để hoàn thành quá trình khởi chạy
- ❖ Ghi lại khoảng thời gian từ thời điểm mà hệ thống đã bắt đầu quá trình khởi chạy cho đến khi khung nhìn đầu tiên của ứng dụng được hiển thị



Các công cụ và phương pháp

Đo thời gian khởi chạy của PWA trên Android :

- ❖ Khi cài đặt PWA trên Android, Chrome sẽ tạo một WebAPK và cài đặt nó.
- ❖ Điều này giúp bạn có thể theo dõi thời gian khởi chạy của ứng dụng theo kiểu gần giống với native Android thông qua Logcat

Công nghệ hiện tại chưa thực hiện được việc đo thời gian khởi chạy của PWA tr

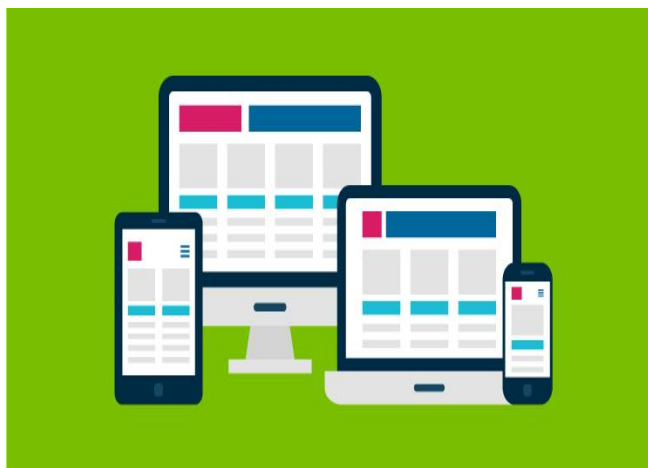
A WebAPK is an APK which is installed by “Add to Home screen” in the Chrome app menu provided that the website meets the [requirements](#).



Các công cụ và phương pháp

Đo thời gian điều hướng

- ❖ Bằng cách ghi lại dấu thời gian ngay sau khi sự kiện chạm bắt đầu điều hướng trang và sau khi trang đích được tạo.
- ❖ Phiên bản native Android được ghi log trong Android Logcat và phiên bản native iOS được ghi log trong Xcode Console
- ❖ Đối với PWA trên Android, sử dụng gỡ lỗi từ xa của Chrome; đối với PWA trên nền tảng iOS, sử dụng Apple Web-inspector của Apple, giúp gỡ lỗi từ xa thông qua Safari.



Các công cụ và phương pháp

Đo kích thước cài đặt

- ❖ Được tính toán sau khi build và gói các ứng dụng để phát hành và được đo bằng megabyte (MB)
- ❖ Trên Android, các ứng dụng được build để phát hành dưới dạng APK. Trên iOS, các ứng dụng được build ở release mode

Câu hỏi: tại sao phải là ở chế độ Release Mode, mà không phải chế độ khác như Debug Mode hoặc Development Mode?



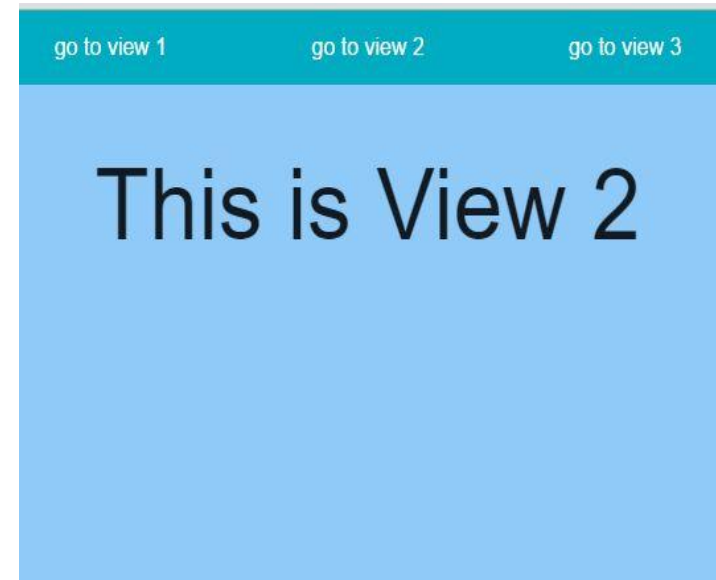
Các thiết bị được chọn

- Chọn thiết bị “cao cấp” và “cấp thấp”
- Các thiết bị đều là vẫn được sử dụng rộng rãi - xét về mặt về phần cứng và phần mềm - vào năm 2017 (đối với cấp thấp) và năm 2020 (đối với cấp cao).

		Android	iOS
High-end	Device	OnePlus 8	iPhone 12
	OS	Android 11	iOS 15.0.1
	CPU	Snapdragon 865	Apple A14 Bionic
	RAM	8GB	4GB
	Released in	2020	2020
	Resolution	1080x2400 pixels	1170x2532 pixels
Low-end	Device	Galaxy S8	iPhone 8
	OS	Android 9	iOS 15.0.1
	CPU	Snapdragon 835	Apple A11 Bionic
	RAM	4GB	2GB
	Released in	2017	2017
	Resolution	1080x2220 pixels	750x1334 pixels

Ứng dụng sẽ được phát triển

- ❖ Ứng dụng được thiết kế để thực hiện công việc này được gọi là “ứng dụng 2 trang” cho phép người dùng điều hướng từ màn hình đầu tiên – được gọi là “Trang chủ” tới màn hình thứ hai – gọi là “Trang A”
- ❖ Màn hình chính có nút bấm ở giữa có thể bắt đầu chuyển hướng đến màn hình thứ hai khi nó bắt được sự kiện chạm.
- ❖ Màn hình thứ hai (Trang A) có văn bản ở giữa màn hình hiển thị tên của nó.
- ❖ Màn hình thứ hai có nút quay lại để cho phép người dùng điều hướng trở lại màn hình đầu tiên.



Mục lục

1. Giới thiệu
2. Các công cụ và phương pháp
3. **Thời gian khởi chạy**
4. Tài nguyên sử dụng
5. Trải nghiệm của đội phát triển
6. Kết luận

Dưới đây là những biểu đồ mô tả sự khác biệt khi app được phát triển bằng các công nghệ khác nhau: công nghệ native, Flutter, React Native, PWA trên các thiết bị Android và iOS ở các phân khúc cao cấp và cấp thấp

I. Thời gian khởi chạy trung bình

Biểu đồ hiển thị cho app ở thiết bị

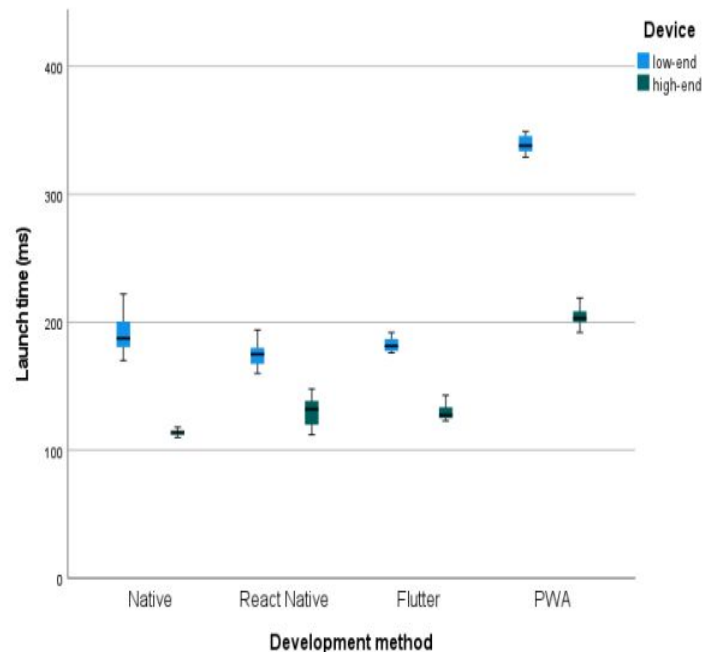
Android cấp thấp:

- Ở React Native: 175 ms (s.d. 9,28 ms), Flutter: 182 ms (s.d. 5,52 ms) và ứng dụng native: 188 ms (s.d. 14,9 ms).

- PWA: 338 ms (s.d. 7,11 ms).

- Kiểm định Mann Whitney U cho thấy sự khác biệt đáng kể giữa React Native với Flutter ($W = 89$, $p = 0,001$) và giữa native với ứng dụng PWA ($W = 0$, $p < 0,001$).

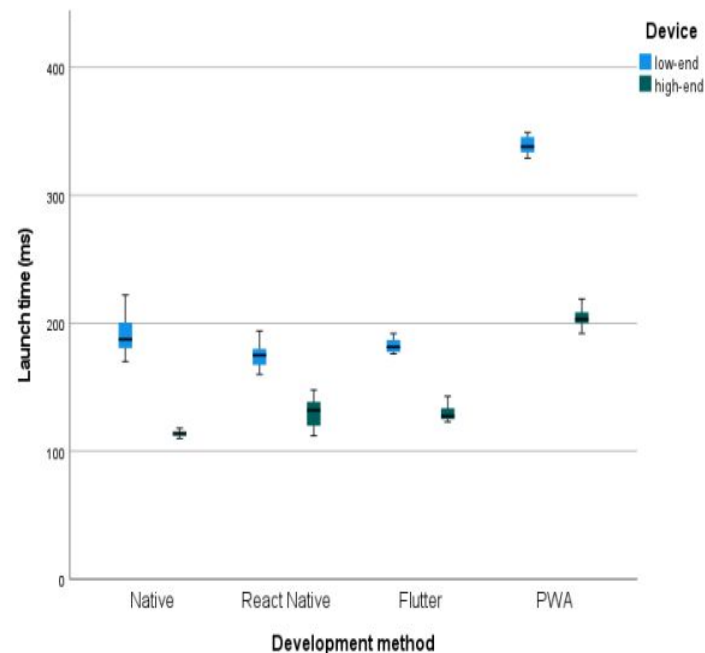
- Ngược lại, không có sự khác biệt đáng kể nào được tìm thấy giữa Flutter với native ($W = 131,5$, $p = 0,0327$).



I. Thời gian khởi chạy trung bình

Kết quả trên thiết bị Android cao cấp có một chút khác biệt:

- Ứng dụng native: 114 ms (s.d. 2,23 ms).
- Các ứng dụng Flutter và React Native là 128 ms (s.d. 5,92 ms) và 132 ms (s.d. 11,3 ms).
- Các ứng dụng PWA là 203 ms (s.d. là 6,56 ms).
- Thử nghiệm cho thấy một sự khác biệt kết quả thời gian khởi chạy giữa bản native và Flutter ($W = 0$, $p < 0,001$) và giữa React Native với PWA ($W = 0$, $p < 0,001$).
- Không có sự khác biệt đáng kể nào được tìm thấy giữa Flutter với React Native ($W = 171,5$, $p = 0,223$).



I. Thời gian khởi chạy trung bình

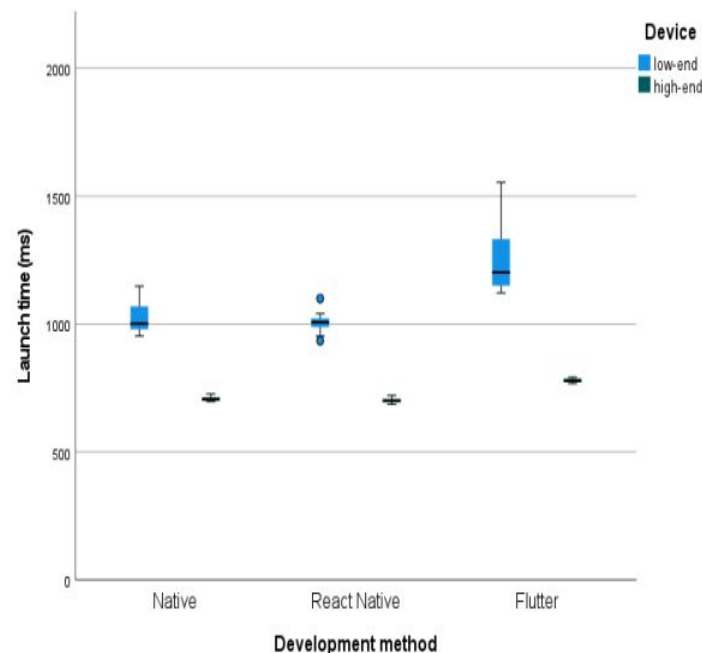
thời gian khởi chạy trung bình trên thiết bị iOS cấp thấp là:

- Native và React Native nhanh nhất lần lượt là 1.002 ms (s.d. 60,5 ms) và 1.006 ms (s.d. 45 ms).

- Flutter chậm hơn: 1202 ms (s.d. 149 ms).

- Sự khác biệt đáng kể về kết quả thời gian khởi chạy trên thiết bị iOS cấp thấp kiểm tra giữa React Native và Flutter ($W = 2$, $p < 0,001$).

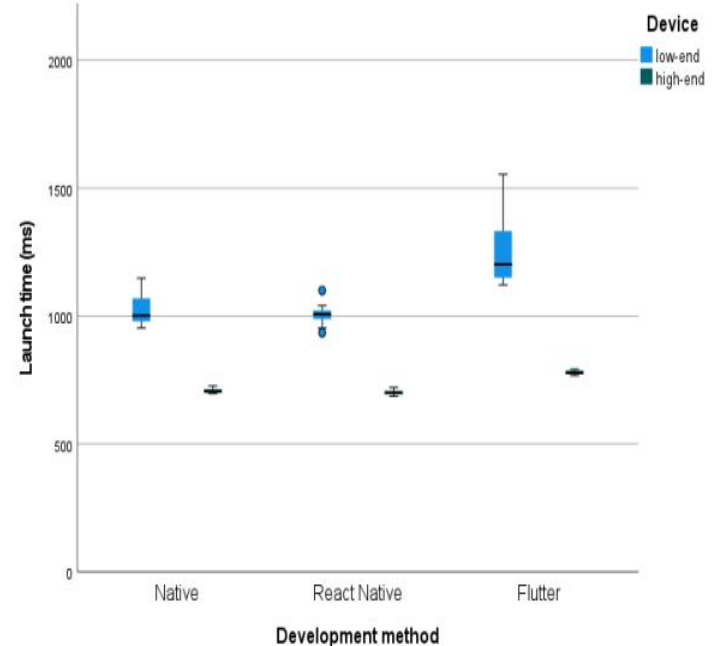
- Ngược lại, không có sự khác biệt đáng kể nào được tìm thấy ứng dụng native và React Native ($W = 221$, $p = 0,719$).



I. Thời gian khởi chạy trung bình

Đối với thiết bị iOS cao cấp, thời gian khởi chạy trung bình cho các phương pháp phát triển được xếp hạng như sau:

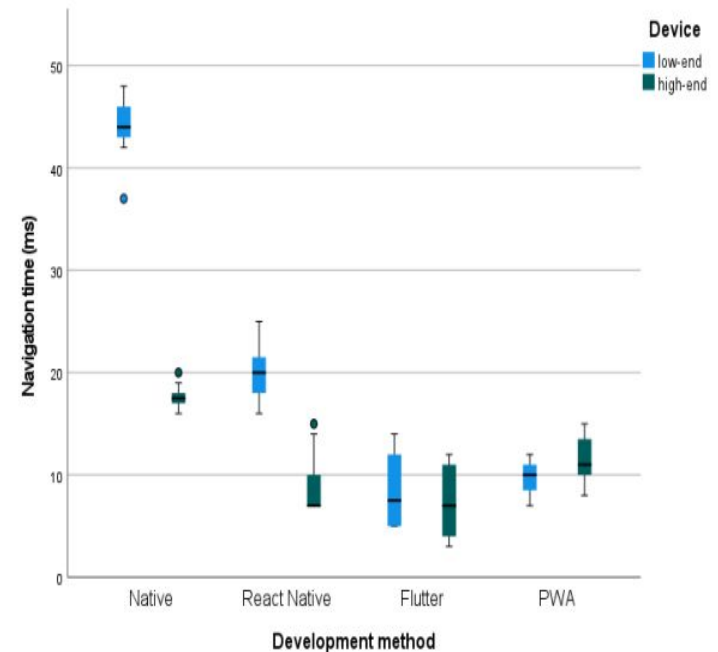
- React Native là nhanh nhất với 700 ms (s.d. 9,46 ms)
- Theo sau là native với 706 ms (s.d. 18,5ms) và cuối cùng, Flutter ở 778 ms (s.d. 13,9 ms).
- Thử nghiệm tiết lộ sự khác biệt có ý nghĩa thống kê giữa React Native và native ($W = 110$, $p = 0,007$) và giữa native với Flutter ($W = 13$, $p < 0,001$).



II. Thời gian điều hướng

Thời gian điều hướng trung bình cho Android “cấp thấp”:

- Flutter với 7,5 ms (s.d. trong tổng số 3,57 ms)
 - PWA với 10 ms (s.d. 1,41 ms).
 - React Native 20 ms (s.d. trong 2,8 ms). Native: 44 ms (s.d. 2,51 ms).
- Sự khác biệt đáng kể giữa PWA và React Native ($W = 0$, $p < 0,001$) và giữa native với React Native ($W = 0$, $p < 0,001$).
- Không có sự khác biệt đáng kể ở Flutter và PWA



II. Thời gian điều hướng

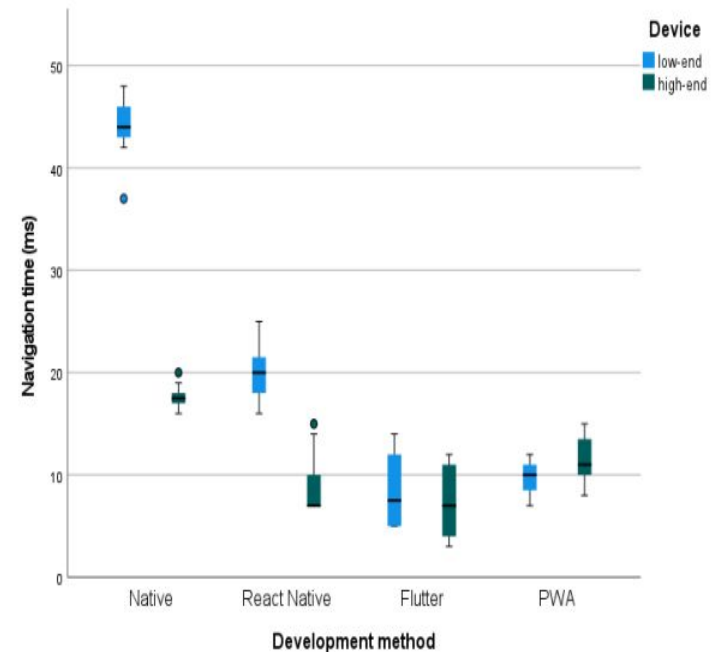
Thời gian điều hướng trung bình cho Android “cao cấp”:

- Flutter và React Native là 7 ms (s.d. là 3,78 ms và 2,83 ms).

- PWA: 11 ms (s.d. là 2,04 ms), còn native là 17,5 ms (s.d. trong 0,93 ms).

- Thử nghiệm cho thấy sự khác biệt đáng kể trên Android cao cấp giữa React Native với PWA ($W = 79,5$, $p < 0,001$) và giữa PWA với native ($W = 0$, $p < 0,001$).

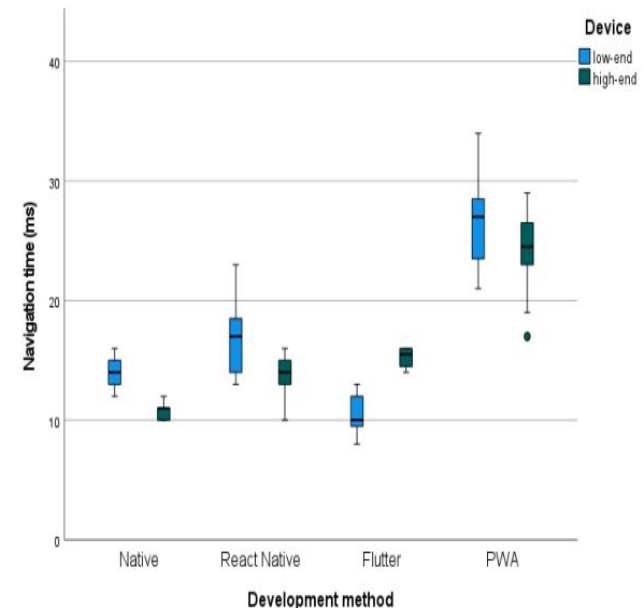
- Không có sự khác biệt giữa Flutter và React Native trên Android cao cấp ($W = 151$, $p = 0,09$).



II. Thời gian điều hướng

Thời gian điều hướng trung bình cho iOS “cấp thấp”:

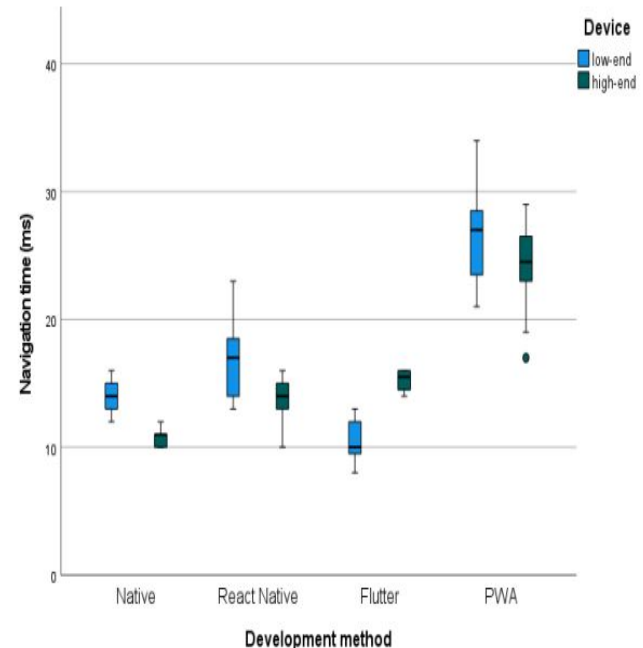
- Flutter: 10 ms (s.d. 1,50 ms).
- iOS native với 14 ms (s.d. trong 1,44 ms), React Native với 17ms (s.d. 2,68 ms).
- PWA: 27 ms (s.d. là 3,28 ms).
- Sự khác biệt đáng kể trên iOS cấp thấp giữa Flutter và native ($W = 21$, $p < 0,001$), giữa bản native và React Native ($W = 72$, $p < 0,001$), cũng như giữa React Native và PWA ($W = 4$, $p < 0,001$).



II. Thời gian điều hướng

Thời gian điều hướng trung bình cho iOS “cao cấp”:

- native: 11 ms (s.d. là 0,74 ms).
- React Native và Flutter là 14 ms (s.d. là 1,92 ms) và 15,5 ms (s.d. là 0,85 ms).
- PWA: 24,5 ms (s.d. 3,08 ms).
- Sự khác biệt đáng kể trên iOS cao cấp giữa native và React Native ($W = 50$, $p < 0,001$), giữa React Native và Flutter ($W = 97,5$, $p = 0,002$) và giữa Flutter và PWA ($W = 0$, $p < 0,001$).



III. Kích cỡ file cài

- Các phương pháp tiếp cận dựa trên PWA và Web: 0,98 MB và 0,71 MB và giống nhau ở cả hai nền tảng
- Trên Android, ứng dụng native: 4,8 MB. Flutter: 16,7 MB. React Native: 32,5 MB
- Trên iOS kích thước thường nhỏ hơn

	Android	iOS
Native	4.8	1.2
React Native	32.5	5
Flutter	16.7	7.9
PWA	0.98	
Web	0.71	

Giải thích kết quả

Tại sao thời gian khởi chạy trung bình của các phương pháp phát triển cho cả hai nền tảng trên các thiết bị cao cấp nhanh hơn đáng kể so với các đối tác cấp thấp của chúng?

- Kết quả thời gian khởi chạy dựa trên phép đo “cold start”, một tình huống khi quy trình của ứng dụng chưa hoạt động (tức là chưa có trong bộ nhớ) và cần được hệ thống tạo lại từ đầu.
- Khởi tạo các thành phần quy trình bên trong của ứng dụng, tạo chế độ xem, tính toán vị trí chế độ xem và đặt chúng xuất hiện trên màn hình nằm trong số các tác vụ đòi hỏi nhiều bộ nhớ và bộ xử lý.
- Ngoài ra, các bước cuối cùng, chẳng hạn như hiển thị các khung nhìn và thực hiện bản vẽ ban đầu, yêu cầu sức mạnh GPU.
- Những thiết bị cấp cao (high-end) được trang bị CPU, memory và SSD hoạt động tốt hơn, do đó giúp chúng đạt được thời gian khởi chạy nhanh hơn đáng kể
- Những điều này, cùng với những thứ khác, đã góp phần giúp các thiết bị cao cấp đạt được thời gian khởi chạy nhanh hơn đáng kể.

Giải thích kết quả

Tại sao thời gian khởi chạy PWA lại chậm hơn đáng kể so với các phương pháp phát triển khác trên Android?

- PWA, không giống như các phương pháp phát triển khác khởi chạy trực tiếp bởi hệ điều hành, dựa vào trình duyệt để bắt đầu.
- Sự phụ thuộc vào một ứng dụng khác (trình duyệt) đã tạo ra một chi phí cao trong quá trình khởi chạy PWA, điều này khiến thời gian khởi chạy của PWA chậm hơn 80,2% trên thiết bị cấp thấp và chậm hơn 53,7% trên thiết bị cao cấp so với phương pháp chậm nhất tiếp theo

Giải thích kết quả

Tại sao thời gian điều hướng của Android native chậm hơn đáng kể so với tất cả các CNDNT khác trên nền tảng Android?

- Điều hướng trong ứng dụng Android gốc dựa trên việc tạo lại “Activity”, một thành phần ứng dụng cấp cao nhất trong Android.
- Trong ứng dụng Android gốc, mỗi trang được lưu trữ bởi một thành phần Activity được tạo lại mỗi khi người dùng điều hướng đến một trang mới. Điều này có nghĩa là khi điều hướng từ trang này sang trang khác, Activity lưu trữ trang đầu tiên sẽ bị hủy và Activity của trang mới cần được tạo.
- Các CNDNT khác áp dụng phương pháp điều hướng được tối ưu hóa hơn bằng cách sử dụng lại thành phần cấp cao nhất.
- Ứng dụng React Native đã sử dụng React Navigation, một thư viện điều hướng dựa trên cộng đồng cho React Native.
- Ngược lại, ứng dụng Flutter không dựa trên giao diện người dùng gốc của nền tảng mà có giao diện người dùng và điều hướng riêng.
- Ứng dụng PWA là một Single Page Application tự động tạo lại một trang theo dữ liệu được chuyển đến trang đó.

Giải thích kết quả

Tại sao kết quả thời gian điều hướng Flutter và PWA thể hiện các tác động khác biệt của loại thiết bị so với các phương pháp phát triển khác?

- Trên Android, từ thiết bị cấp thấp đến cao cấp cho các native và React Native đã giảm thời gian điều hướng lần lượt là 60,2% và 65%;

Trên Flutter chỉ giảm 6,6%, PWA: tăng 10%.

- Trên iOS từ thiết bị cấp thấp sang thiết bị cao cấp, thời gian điều hướng giảm 21,4% và 17,6% đối với iOS native và React Native,

chỉ giảm 9,2% đối với PWA, tăng 5,5% với Flutter.

Điều hướng trong các native và React Native dựa trên các API do OS - Activity và Fragment và UINavigationController

Flutter dựa trên điều hướng của riêng nó và điều hướng của PWA dựa trên trình duyệt.

- iOS cấp cao mức độ giảm thời gian điều hướng ít hơn so với Android cấp cao: iOS cấp cao tăng tới 196% số pixel.

Ở Android là tăng 8.1% số pixel

Giải thích kết quả

Tại sao kích thước tệp trình cài đặt cho PWA và native lại nhỏ hơn kích thước tệp cho React Native và Flutter?

- Các native được build bởi bộ phát triển phần mềm (SDK), các công cụ và ngôn ngữ lập trình do nền tảng của chúng cung cấp thường sử dụng các framework, thư viện và API đã có sẵn trên các nền tảng tương ứng của chúng. Do đó, kích thước của những ứng dụng này có xu hướng tương đối nhỏ
- Các CNDNT này cũng được đóng gói thêm các thư viện, framework và API.
- PWA tận dụng những gì đã có của trình duyệt.

Giải thích kết quả

Ngưỡng nào cho thời gian khởi động

- Giới hạn cho các hoạt động trong đó luồng suy nghĩ của người dùng không bị gián đoạn là 1,0 giây, mặc dù có thể nhận thấy độ trễ.
- Như vậy thời gian khởi động không nên chậm hơn một giây

Giải thích kết quả

Khả năng đáp ứng nhận được từ kết quả của thời gian điều hướng

- Giới hạn thời gian để phản hồi của hệ thống có thể được người dùng chấp nhận là 0,1 giây.

- Thời gian điều hướng chậm nhất:

 - trên Android cấp thấp của native: 44 ms

 - trên iOS cấp thấp của PWA: 27 ms

Giải thích kết quả

❖ *Kết quả tổng thể cho từng phương pháp phát triển*

- iOS native cung cấp khả năng đáp ứng tổng thể tốt nhất.
- Kết quả tổng thể cho các ứng dụng Flutter và React Native rất giống nhau. Chúng lựa chọn thuận lợi nhất trong Android và đứng thứ 2 trong iOS.
- Kết quả tổng thể của Android native kém React Native và Flutter nhưng tốt hơn PWA.
- PWA kém nhất.

Mục lục

1. Giới thiệu
2. Các công cụ và phương pháp
3. Thời gian khởi chạy
4. **Tài nguyên sử dụng**
5. Trải nghiệm của đội phát triển
6. Kết luận

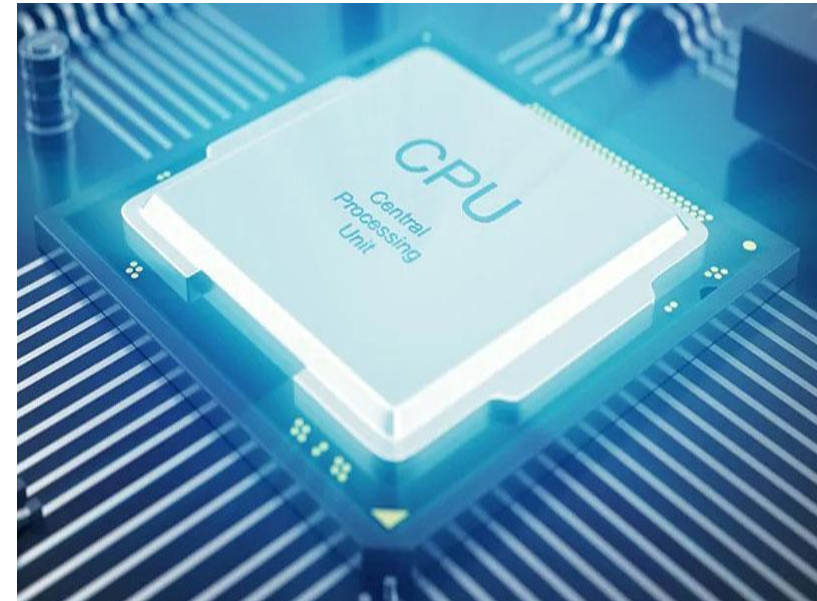
Tiêu chí và chỉ số đánh giá



I. CPU Usage

Mức sử dụng CPU (CPU usage)

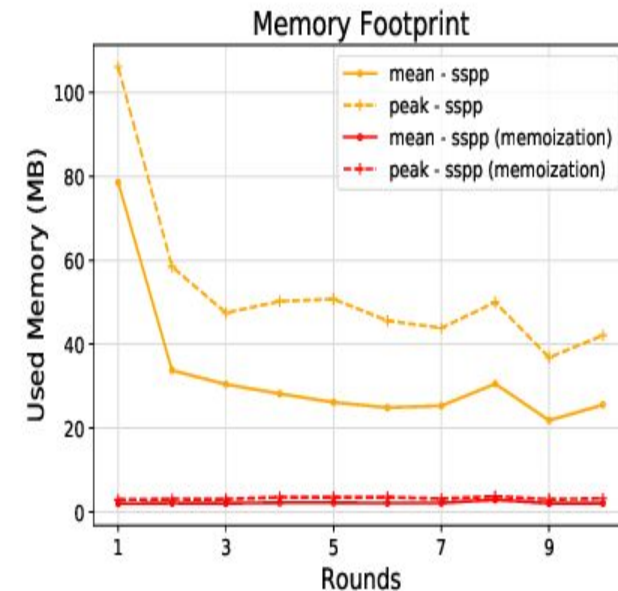
- Là phần trăm tổng dung lượng CPU tại một thời điểm nhất định
- Mức sử dụng CPU là một số liệu để định lượng tải của bộ xử lý bằng cách chạy các chương trình máy tính



II. Memory Footprint

Mức chiếm dụng bộ nhớ (Memory footprint)

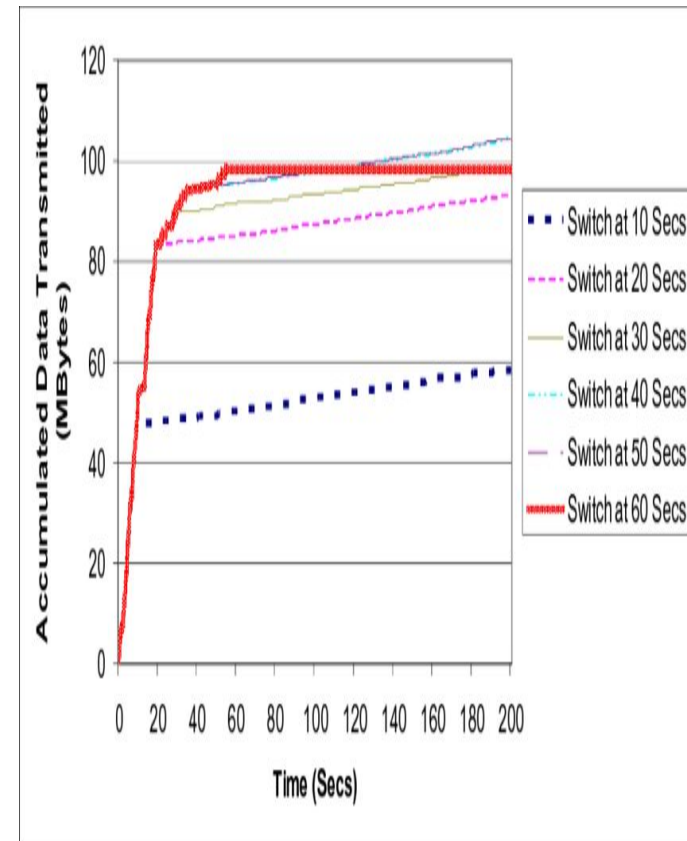
- Là lượng bộ nhớ chính mà một chương trình sử dụng hoặc tham chiếu trong khi chạy.
- CPU dựa vào bộ nhớ để đọc và cung cấp dữ liệu để xử lý.
- Mỗi chương trình sẽ yêu cầu phân bổ một lượng bộ nhớ nhất định để chạy đúng cách.
- Do đó mức chiếm dụng bộ nhớ của một ứng dụng là một yếu tố chính trong việc đánh giá.



III. Total data transferred

Tổng dữ liệu được truyền (Total data transfered)

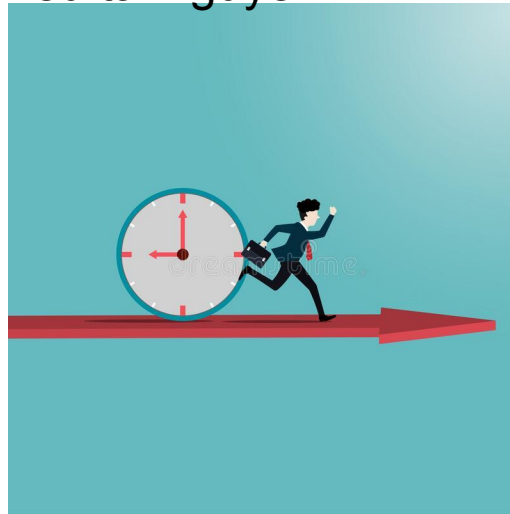
- Là tổng dữ liệu được nhận và gửi qua mạng trong một khoảng thời gian cụ thể, được đo bằng megabyte (MB)
- Hầu hết các thiết bị di động ngày nay được kết nối liên tục với mạng, qua đó chúng gửi và nhận dữ liệu.
- Điều này đặc biệt quan trọng đối với điện thoại có ứng dụng hướng dữ liệu thường được kết nối với máy chủ từ xa



IV. Execution time

thời gian thực thi (Execution time)

- Là khoảng thời gian trôi qua từ khi bắt đầu kiểm tra sử dụng nhiều tài nguyên đến khi hoàn thành



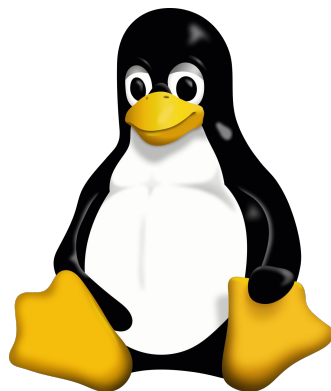
Các công cụ và phương pháp



I. CPU usage measurement

❖ Đo mức sử dụng CPU trên Android

- Áp dụng cho React Native, Flutter và Android native và PWA
- Dùng lệnh top của Linux, để báo cáo phần trăm sử dụng CPU
- Dùng Android Debug Bridge (ADB)



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5752	root	20	0	176m	25m	12m	S	6	0.4	3:53.01	jsvc

*PID(5752):	PID of running Process
*USER(root):	User under which process is running
*PR(20):	Priority Of running Process
*NI(0):	Nice Value of running Process
*VIRT(176):	Virtual Memory used by Process
*RES(25m):	Physical Memory used by Process
*SHR(12m):	Shared Memory used by Process
*S(S):	Current Status of Running Process
*%CPU(6):	% CPU Used by this Processes
*%MEM(0.4):	% RAM Used by this Process
*TIME+(3:53:01):	Total time of process running for
*COMMAND(jsvc):	Name of Process

I. CPU usage measurement

- ❖ Đo mức sử dụng CPU trên iOS
 - Có thể cho các native, React Native và Flutter bằng “Instrument” của Xcode.
 - PWA hiện tại không thể theo dõi được.



II. Memory footprint measurement

Việc đo dung lượng bộ nhớ được thực hiện bởi các công cụ đo lường tương tự như việc sử dụng CPU:

- Trên Android, chạy ứng dụng trên Android-Top command, liệt kê các số liệu liên quan đến bộ nhớ.
- Trên iOS, với native iOS, React Native và Flutter, “Allocations” trong Instrument theo dõi tổng bộ nhớ.
- Đối với PWA trên iOS sử dụng Safari Web-inspector,

Mức sử dụng bộ nhớ tính bằng megabyte (MB) được lấy mẫu sau mỗi 500 ms.

Giá trị trung bình của tất cả các mẫu trong một thử nghiệm là mức chiếm dụng bộ nhớ trung bình



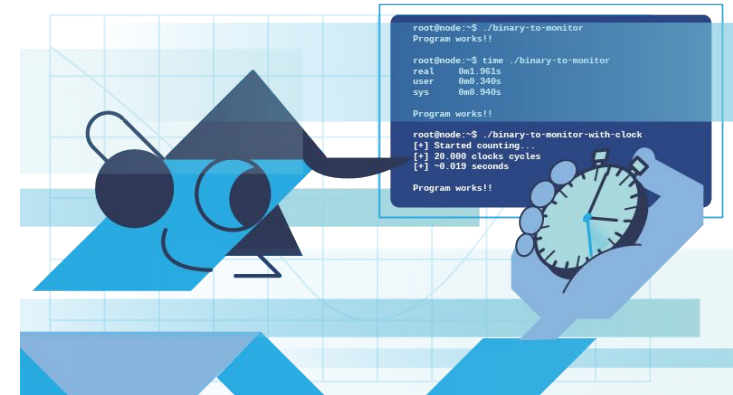
III. Total data transferred

- ❖ Trên Android, đối với các native, React Native và Flutter, Android Studio's Profiler đánh giá tổng dữ liệu được gửi và nhận qua mạng.
- ❖ Đối với các ứng dụng PWA chạy trên Android và dựa vào gỡ lỗi từ xa của Chrome để đo tổng dữ liệu được truyền từ tab “Mạng”.
- ❖ Trên iOS, đối với các native apps, React Native và Flutter, “Kết nối mạng” trong Instrument đã báo cáo tổng dữ liệu đã truyền và nhận.
- ❖ Để đo tổng dữ liệu được truyền cho PWA chạy trên iOS, tab “Mạng” của trình kiểm tra Web Safari đã được sử dụng.



IV. Execution time

- ❖ Thời gian thực thi của các phiên bản gốc Android đã được ghi vào Android logcat.
- ❖ Các phiên bản native của iOS đã được ghi vào trong Bảng điều khiển Xcode.
- ❖ Đối với PWA trên Android, chúng tôi đã sử dụng gỡ lỗi từ xa của Chrome để theo dõi bảng điều khiển.
- ❖ Với PWA trên iOS, các nhà nghiên cứu đã tận dụng trình kiểm tra Web của Apple, giúp gỡ lỗi từ xa thông qua Safari.



Các thiết bị được chọn



Thiết bị được lựa chọn

- Thực hiện đo trên các thiết bị cấp cao (high-end) và cấp thấp (low-end)
- Ứng dụng được thiết kế là một ứng dụng một trang có ba nút.
- Mỗi nút bắt đầu một resource-intensive test và ghi lại thời gian thực thi kiểm tra.
- Tất cả các ứng dụng được phát triển với cấu hình mặc định của chúng, ngôn ngữ lập trình thông thường nhất và sử dụng IDE được đề xuất
- Các bài test: CPU-intensive test, memory-intensive test, network-intensive test

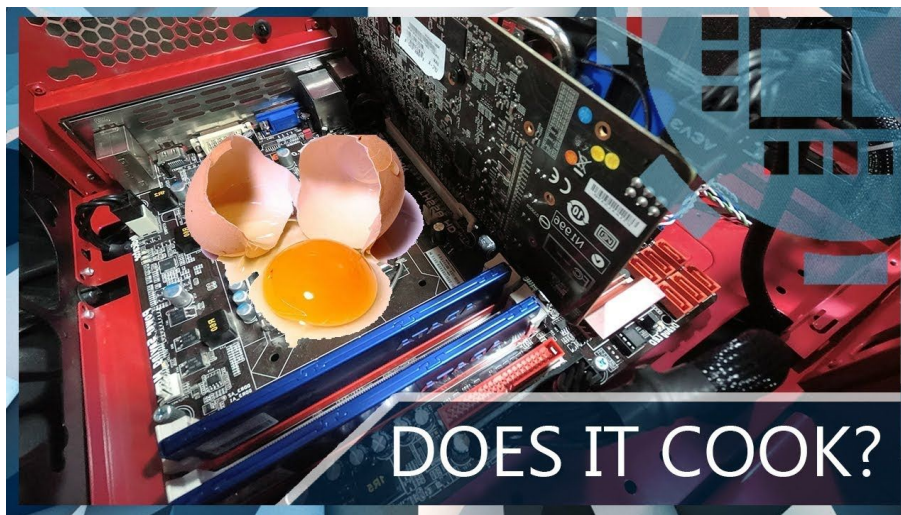
		Android	iOS
High-end	Device	OnePlus 8	iPhone 12
	OS	Android 11	iOS 15.0.1
	CPU	Snapdragon 865	Apple A14 Bionic
	RAM	8GB	4GB
	Released in	2020	2020
	Resolution	1080x2400 pixels	1170x2532 pixels
Low-end	Device	Galaxy S8	iPhone 8
	OS	Android 9	iOS 15.0.1
	CPU	Snapdragon 835	Apple A11 Bionic
	RAM	4GB	2GB
	Released in	2017	2017
	Resolution	1080x2220 pixels	750x1334 pixels

Design and implementation of apps



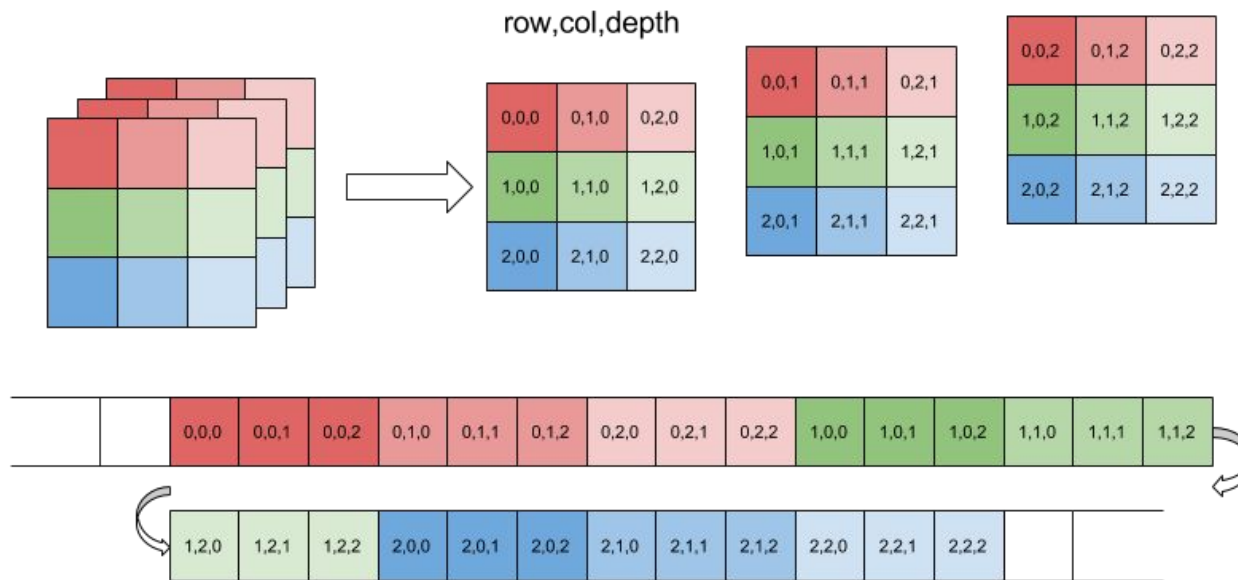
I. CPU-intensive Test

- ❖ Một bài kiểm tra chuyên sâu về CPU để đánh giá việc sử dụng tài nguyên và hiệu suất, đặc biệt tập trung vào việc sử dụng CPU của ứng dụng.
- ❖ Bài kiểm tra này được thiết kế để tìm các số Nguyên tố
- ❖ Việc triển khai thử nghiệm nhận một số nguyên “N” và xác định tất cả các số Nguyên tố từ 1 đến “N”



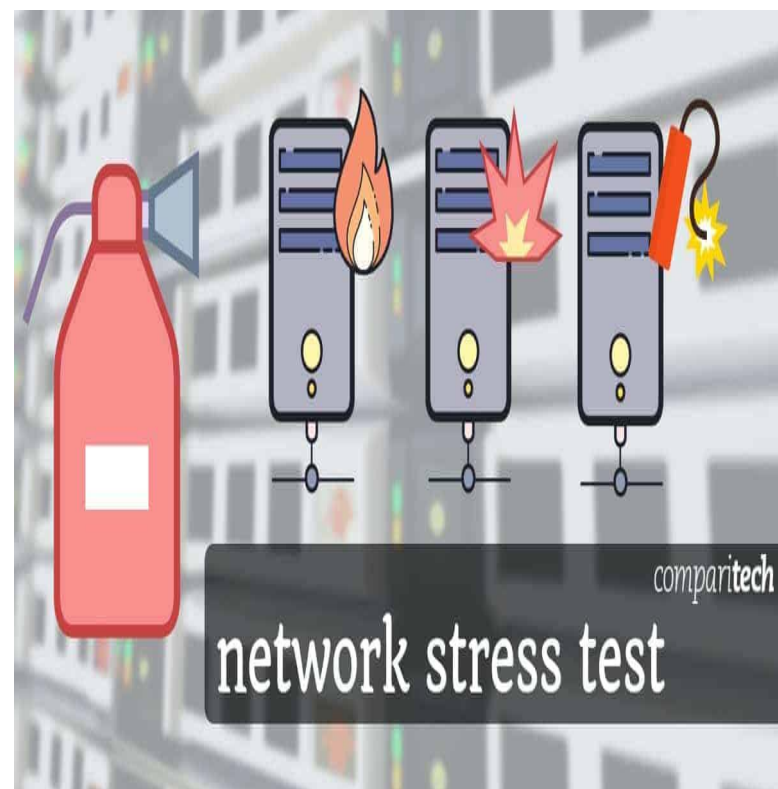
II. Memory-intensive Test

- ❖ Bài kiểm tra sử dụng nhiều bộ nhớ: tìm tất cả chu trình của một đồ thị lớn
- ❖ Việc triển khai thuật toán Floyd-Warshall đã được sử dụng để lấy ma trận liên kề. Thử nghiệm đã được tùy chỉnh để đẩy mức tiêu thụ bộ nhớ đến giới hạn của nó và đồng thời, có thể hoàn thành.
- ❖ Chỉ có một phương pháp phát triển - PWA trên thiết bị iOS cấp thấp - không thể hoàn thành bài kiểm tra.



III. Network-intensive Test

- ❖ Một bài kiểm tra chuyên sâu về mạng được thiết kế để đánh giá việc sử dụng tài nguyên và hiệu suất của một ứng dụng trong các hoạt động mạng chuyên sâu và truyền dữ liệu qua mạng.
- ❖ Bài kiểm tra dựa trên giao tiếp và truyền dữ liệu giữa các ứng dụng và back-end thông qua RESTful API
- ❖ Ứng dụng iOS native được build bằng ngôn ngữ lập trình Swift .





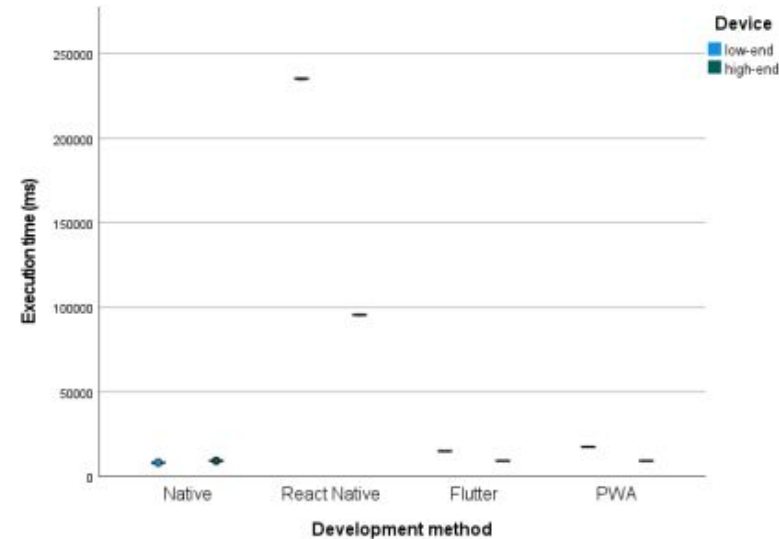
RESULTS

Tổng kết

I. CPU-Intensive test

thời gian thực thi trên Android

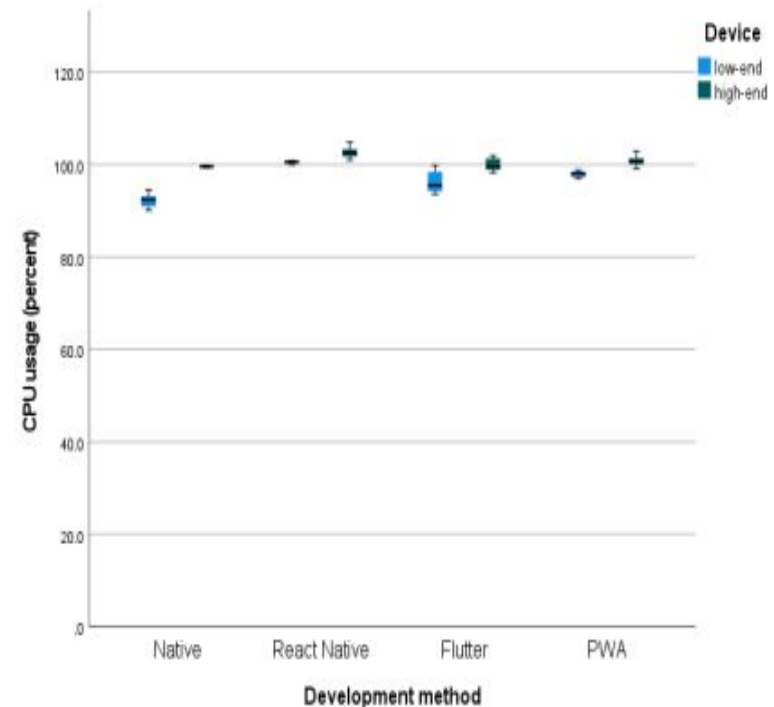
- Đối với thiết bị Android cấp thấp, thời gian thực thi trung bình cho native: 8.100,5 ms (s.d. 15,88 ms), Flutter: 14.901,5 ms (s.d. 16,3 ms), PWA: 17.494,5 ms (s.d. 38,97 ms). React Native với 235.223 ms (s.d. 265,01 ms)
- Thiết bị Android cao cấp: PWA: 9.206 ms (s.d. 4,65 ms), Flutter: 9.219 ms (s.d. 489,93 ms), và native: 9.230 ms (s.d. 11,96 ms), React Native: 95.471,5 ms (s.d. là 137,37 ms)



I. CPU-Intensive Test

Mức sử dụng CPU trên Android

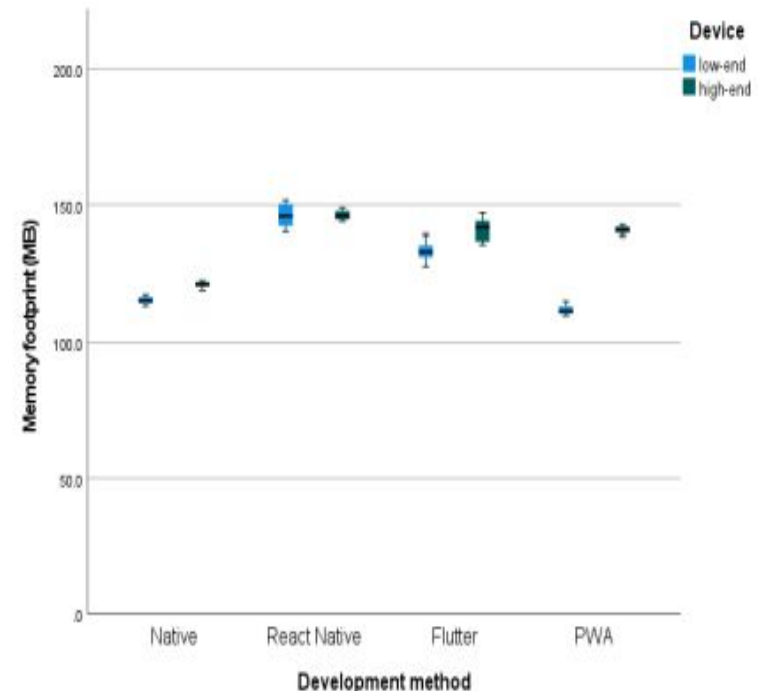
- Android cấp thấp, native ở 92,4% (s.d. 1,30%), Flutter ở 95,5% (s.d. 2,26%), PWA là 98,0% (s.d. 0,62%), React Native ở 100,5% (s.d. 0,25%)
- Thiết bị Android cấp cao, Flutter ở 99,5% (s.d. 1,28%), native ở 99,6% (s.d. 0,14%), PWA là 100,6% (s.d. 1,01%), React Native ở 102,5% (s.d. 1,18%)



I. CPU-Intensive Test

Mức chiếm dụng bộ nhớ trên Android

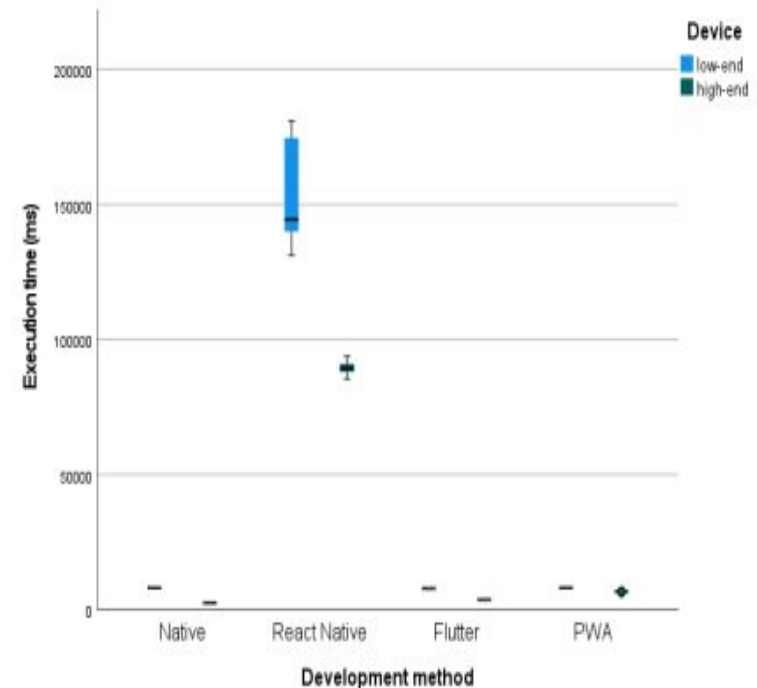
- Trên thiết bị Android cấp thấp, PWA: 111,6 MB (s.d. 1,74 MB), native ở 115,7 MB (s.d. 1,38 MB) và Flutter ở 133,5 MB (s.d. 3,10 MB). React Native 145,9 MB (s.d. 3,95 MB).
- Thiết bị Android cao cấp, native có 121,5 MB (s.d. 1,08 MB), PWA ở 140,7 MB (s.d. 1,25 MB) và Flutter ở 141,8 MB (s.d. 3,73 MB). React Native: 145,9 MB (s.d. 1,54 MB)



I. CPU-Intensive Test

Thời gian thực thi trên iOS

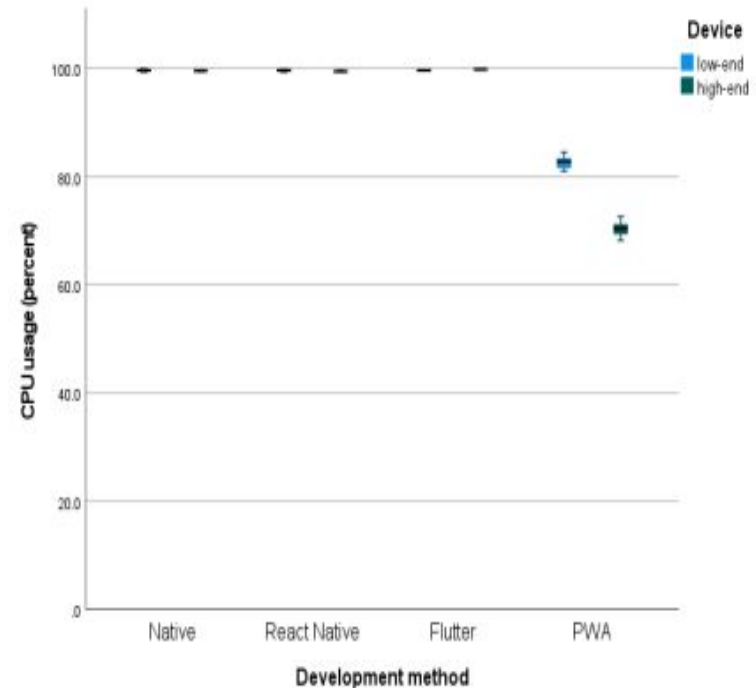
- Trên thiết bị iOS cấp thấp, Flutter: 7874 ms (s.d. 13,05 ms), PWA 8136 ms (s.d. 5,88 ms) và native ở 8140,5 ms (s.d. 9,09 ms). React Native là chậm nhất để hoàn thành, với 144525,5 ms (s.d. 18133,76 ms)
- iOS cao cấp, native: 2537 ms (s.d. 1,7 ms), Flutter ở 3768,5 ms (s.d. 38,22 ms), PWA ở 6718,5 ms (s.d. 45,51 ms), React Native ở 89284 ms (s.d. 2141,37 ms)



I. CPU-Intensive Test

Mức sử dụng CPU trên iOS

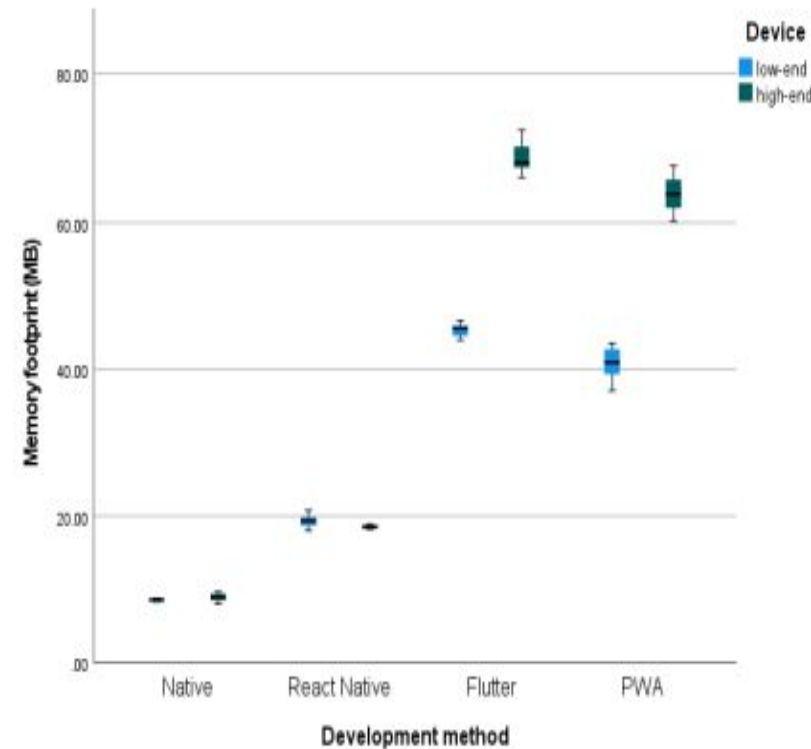
- Thiết bị iOS cấp thấp, PWA: 82,7% (s.d 1,13%), native ở mức 99,6% (s.d 0,21%), Flutter ở 99,6% (s.d 0,08%) và React Native ở 99,6% (s.d 0,15%)
- Thiết bị cao cấp iOS, PWA: 70,4% (s.d. 1,33%), React Native ở 99,4% (s.d. 0,08%), native ở 99,5% (s.d. 0,07%) và Flutter ở 99,8% (s.d. 0,06%)



I. CPU-Intensive Test

Mức chiếm dụng bộ nhớ trên iOS

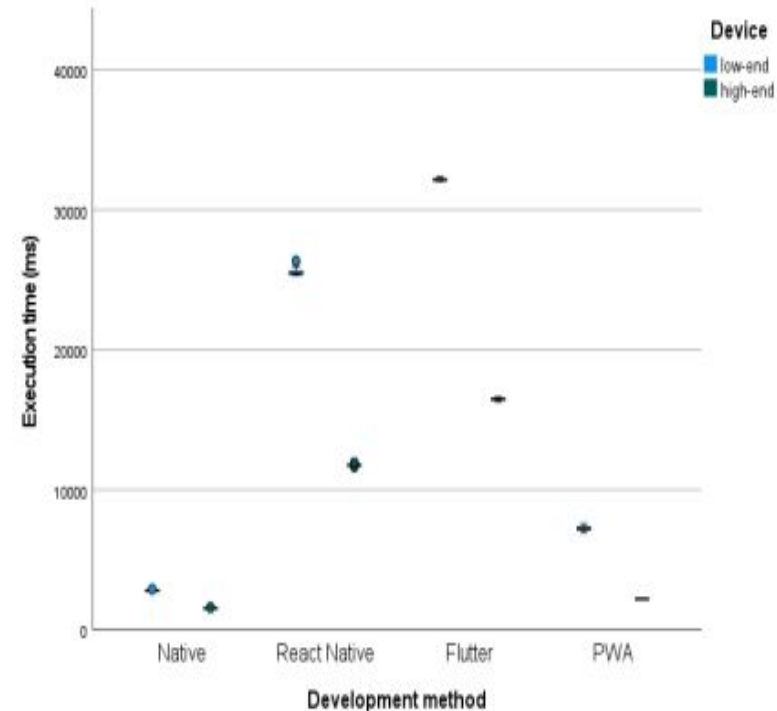
- Với thiết bị iOS cấp thấp, native: 8,6 MB (s.d. 0,19 MB), React Native với 19,3 MB (s.d. 0,85 MB). PWA và Flutter 40,9 MB (s.d. là 2,19 MB) và 45,6 MB (s.d. là 0,86 MB)
- Trên thiết bị iOS cao cấp, native: 9,0 MB (s.d. 0,53 MB), React Native ở 18,5 MB (s.d. 0,25 MB), PWA ở 63,9 MB (s.d. 2,24 MB) và Flutter ở 68,2 MB (s.d. 1,89 MB)



II. Memory-Intensive Test

Thời gian thực thi trên Android

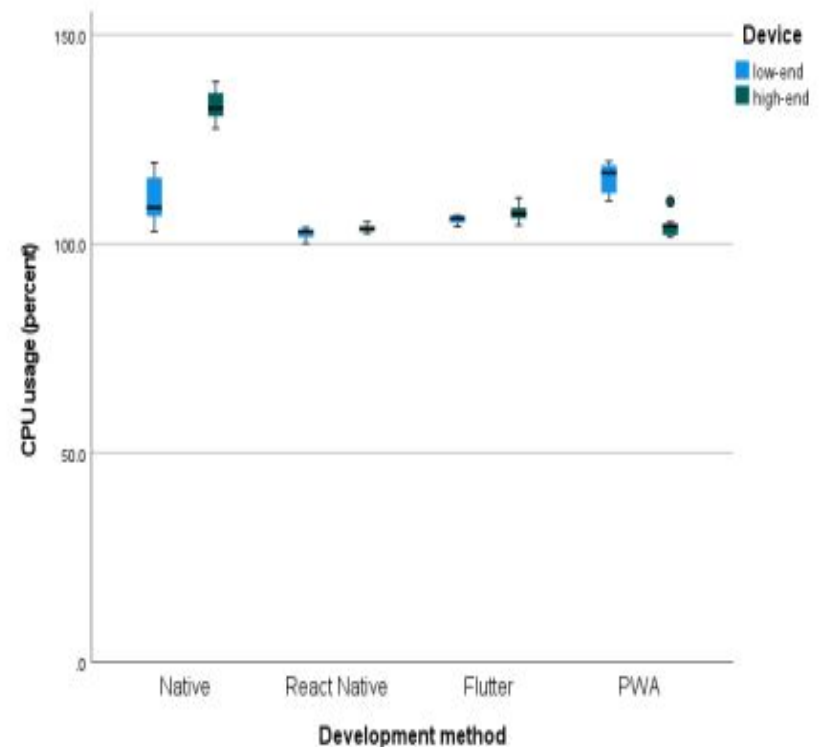
- Thiết bị Android cấp thấp, native là thấp nhất ở 2804,5 ms (s.d. 36,77 ms), PWA ở 7233 ms (s.d. 181,32 ms), React Native ở 25465 ms (s.d. 260,16 ms), Flutter với 32155 ms (s.d. 90,85 ms)
- Trên thiết bị cao cấp, native Android nhanh nhất, với 1530 ms (s.d. 48,39 ms), PWA ở 2203 ms (s.d. 134,38 ms) và React Native ở 11763 ms (s.d. 127,31 ms). Flutter: với 6487 ms (s.d. 96,56 ms)



II. Memory-Intensive Test

Mức sử dụng CPU trên Android

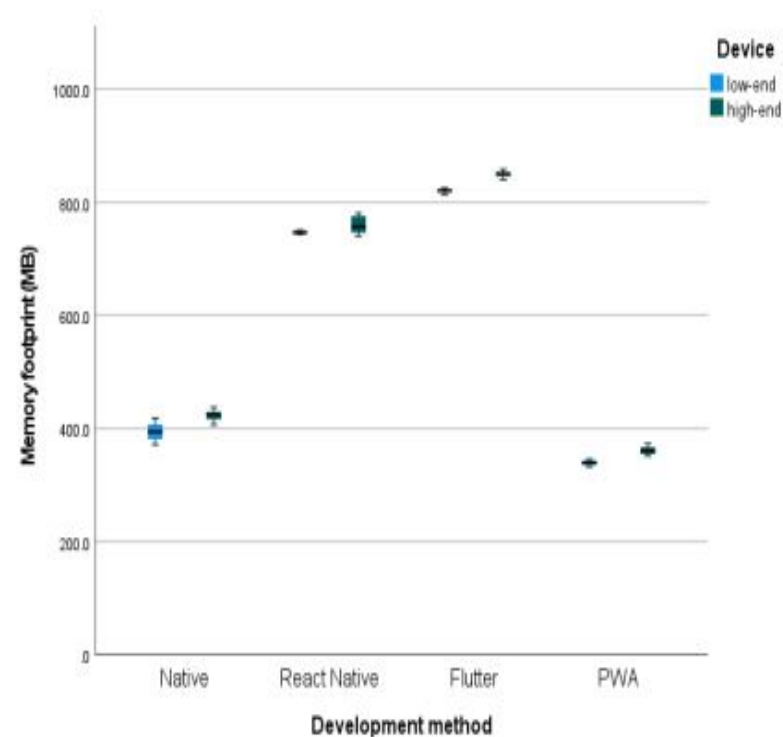
- Với thiết bị Android cấp thấp, React Native thấp nhất là 102,9% (s.d. 1,10%), Flutter 106,1% (s.d. 0,93%), native 108,6% (s.d. 5,37%), PWA với 116,9% (s.d. 3,51%)
- Trên thiết bị Android cao cấp, React Native: 103,6% (s.d. 0,81%), PWA 104,3% (s.d. 1,93%), Flutter 107,1% (s.d. 1,85%), native 132,4% (s.d. là 3,43%)



II. Memory-Intensive Test

Mức chiếm dụng bộ nhớ trên Android

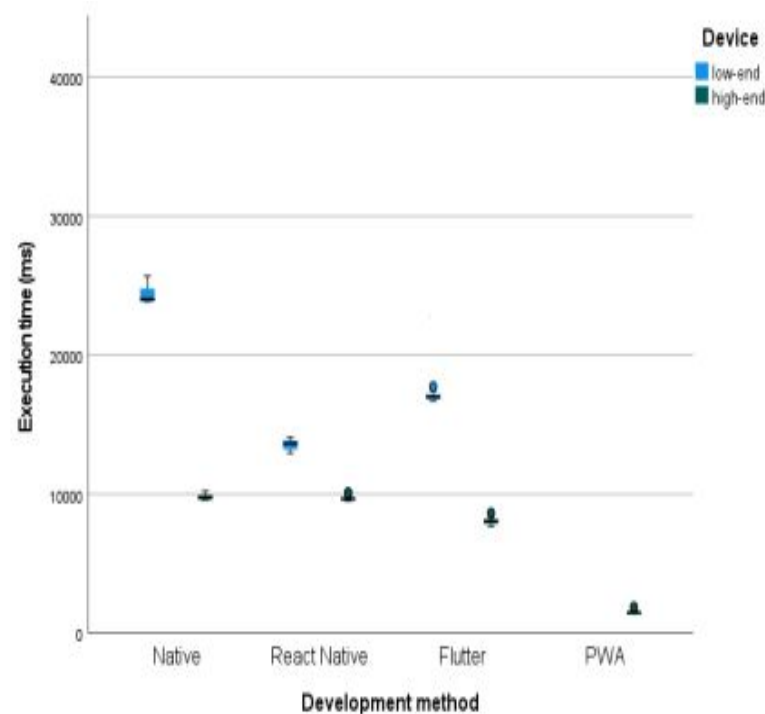
- Với thiết bị Android cấp thấp, PWA: 339,2 MB (s.d. trong số 4,62 MB), 394,1 MB (s.d. 15,98 MB), React Native và Flutter: 746,5 MB (s.d. 2,33 MB) và 820,4 MB (s.d. 2,94 MB)
- Trên thiết bị Android cao cấp, PWA: 360,2 MB (s.d. 7,21 MB), native với 424,9 MB (s.d. 8,61 MB). Sau cùng là React Native và Flutter với 756,7 MB (s.d. 14,43 MB) và 850,3 MB (s.d. 4,70 MB)



II. Memory-Intensive Test

Thời gian thực thi trên iOS

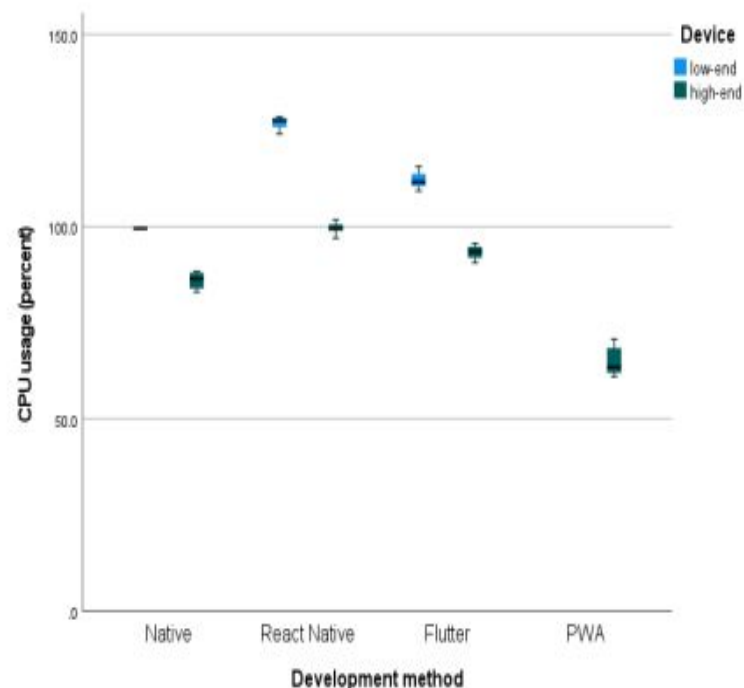
- Với thiết bị iOS cấp thấp, React Native 13.659 ms (s.d. 379,44 ms), Flutter 17.012 ms (s.d. 1298,37) và native 24.026,5 ms (s.d. 1462,96ms)
- Trên thiết bị cao cấp iOS, PWA: 1.460 ms (s.d. 107,44 ms), Flutter 8.072,5 ms (s.d. 222,42 ms), React Native 9.651 ms (s.d. 158,52 ms) và native 9.775 ms (s.d. 200,41 ms)



II. Memory-Intensive Test

Mức sử dụng CPU trên iOS

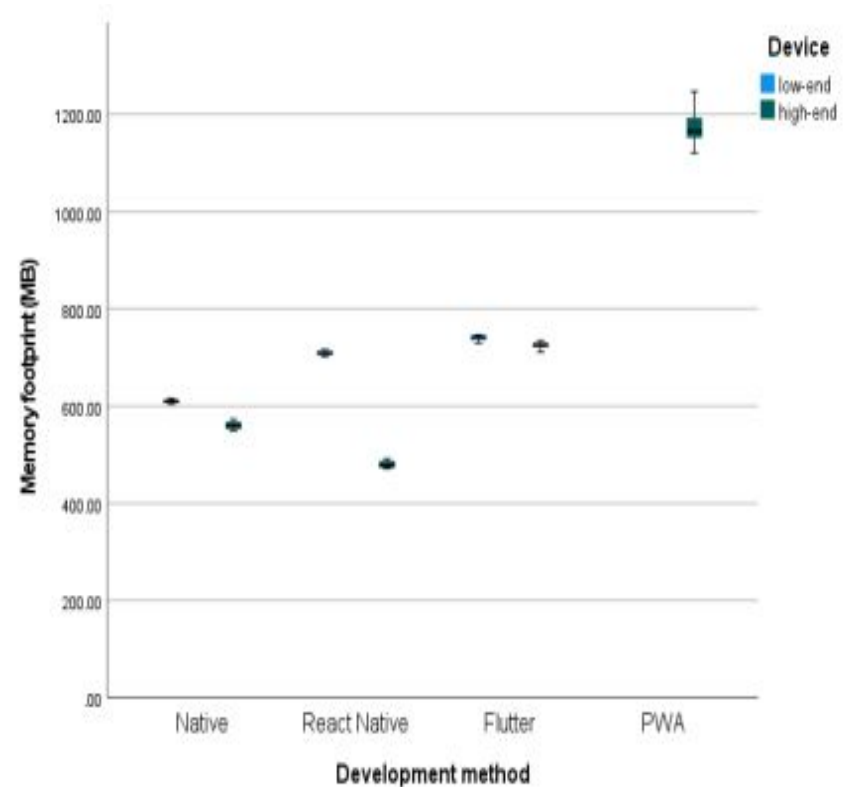
- Với thiết bị iOS cấp thấp, native: 99,5% (s.d. 0,07%), Flutter 111,6% (s.d. 2,06%) và React Native 127,6% (s.d. 1,43%)
- Trên thiết bị cao cấp iOS, PWA: 63,5% (s.d. là 3,48%), sau đó là native 86,4% (s.d. 1,95%), mức sử dụng của Flutter và React Native là 93,5% (s.d. 1,69%) và 99,5% (s.d. 1,32%).



II. Memory-Intensive Test

Mức chiếm dụng bộ nhớ trên iOS

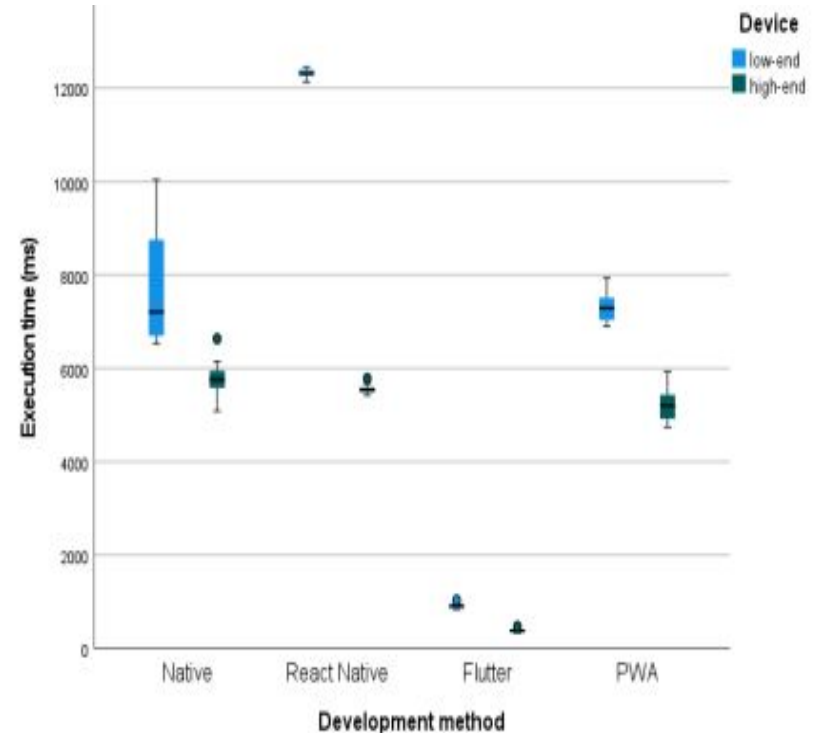
- Với thiết bị iOS cấp thấp, native: 610,0 MB (s.d. 3,89 MB). React Native và Flutter là 709,2 MB (s.d. 4,58 MB) và 742,1 MB (s.d. 5,13 MB)
- Trên thiết bị cao cấp iOS, native: 559 MB (s.d. là 8,30 MB), React Native 478,2 MB (s.d. 7,30 MB), Flutter 725,5 MB (s.d. 6,47 MB), PWA 1165,7 MB (s.d. 32,44 MB)



III. Network-Intensive Test

thời gian thực thi trên Android

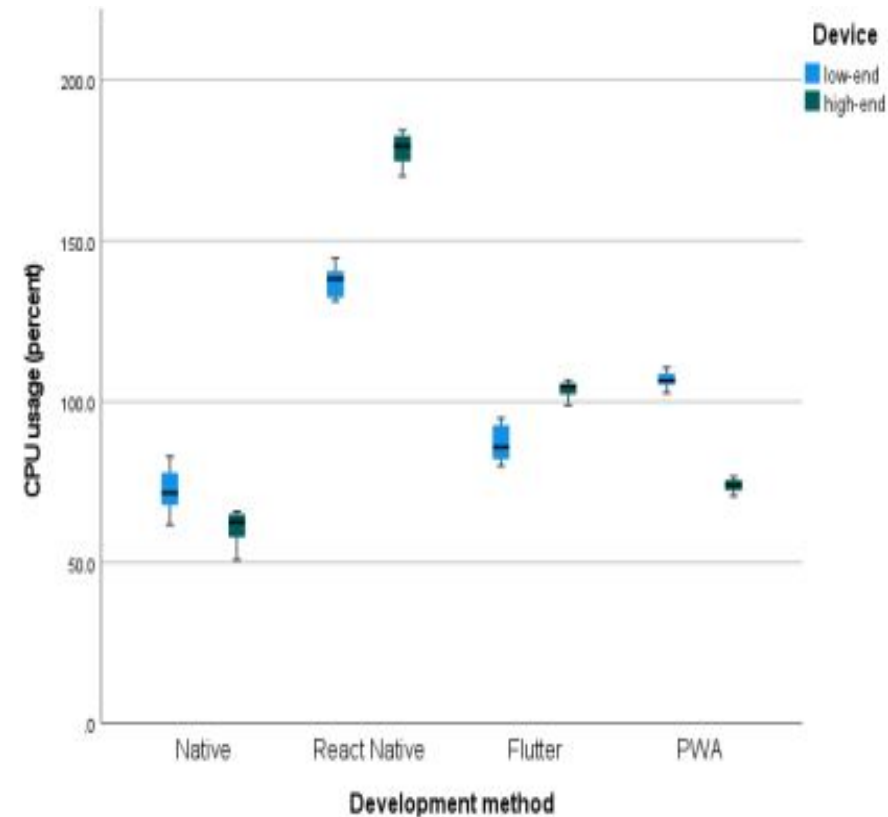
- Với thiết bị Android cấp thấp, Flutter: 914 ms (s.d. 46,33 ms). của Native và PWA với 7207,5 ms (s.d. 1142,82 ms) và 7296 ms (s.d. 307,70 ms). React Native chậm với 12312,5 ms (s.d. 85,82 ms)
- Trên thiết bị Android cao cấp, Flutter: 380 ms (s.d. là 31,53 ms), PWA 5200 ms (s.d. 328,09 ms), React Native 5544,5 ms (s.d. 85,04 ms) và native 5761 ms (s.d. 345,45 ms)



III. Network-Intensive Test

Mức sử dụng CPU trên Android

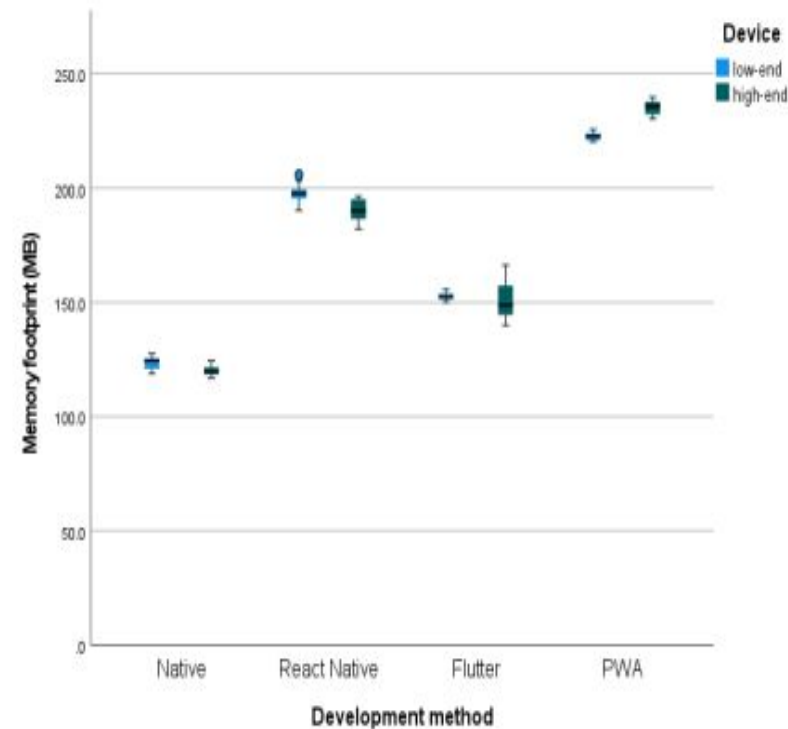
- Với thiết bị Android cấp thấp, native: 71,6% (s.d. 6,23%), Flutter 85,9% (s.d. 5,36%), PWA 106,5% (s.d. 2,43%) và React Native 64 138,2% (s.d. 4,54%)
- Trên thiết bị Android cao cấp, native: 62,5% (s.d. 4,40%), tiếp là PWA 74% (s.d. 1,95%), Flutter 104,8% (s.d. 2,32%) và React Native 179,6% (s.d. 4,47%)



III. Network-Intensive Test

Mức chiếm dụng bộ nhớ trên Android

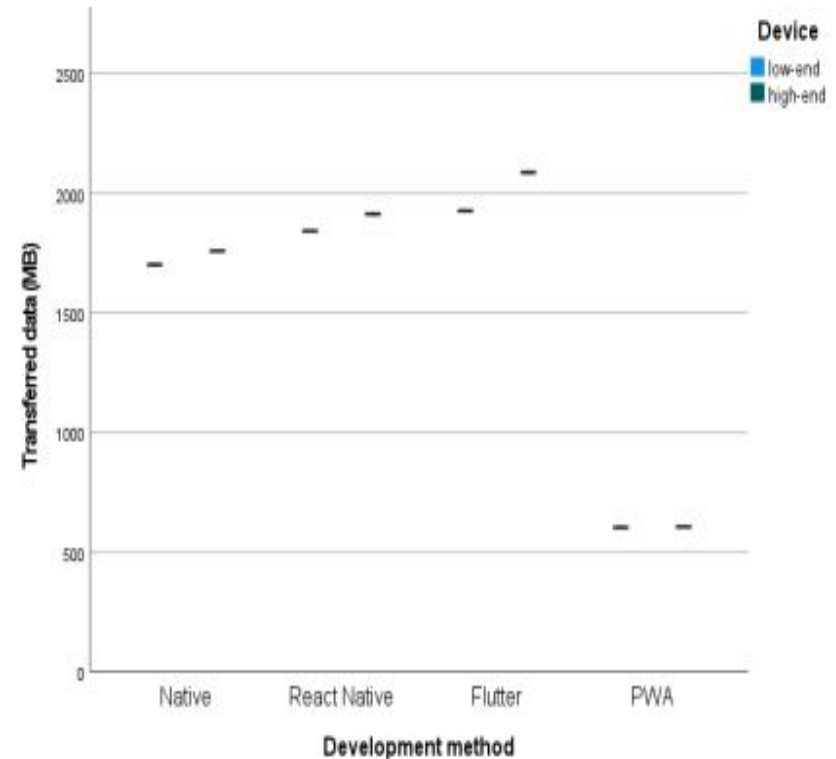
- Với thiết bị Android cấp thấp, native: 124,5 MB (s.d. 2,70 MB), Flutter 152,3 MB (s.d. 1,47 MB), React Native 197,6 65 MB (s.d. 3,95 MB) và PWA 222,7 MB (s.d. 1,64 MB)
- Trên thiết bị Android cao cấp, native: 119,6 MB (s.d. 2,05 MB), Flutter 148,8 MB (s.d. 7,85 MB), React Native 190,1 MB (s.d. 4,71 MB) và PWA 235,7 MB (s.d. 3,05 MB)



III. Network-Intensive Test

Dữ liệu được truyền trên Android

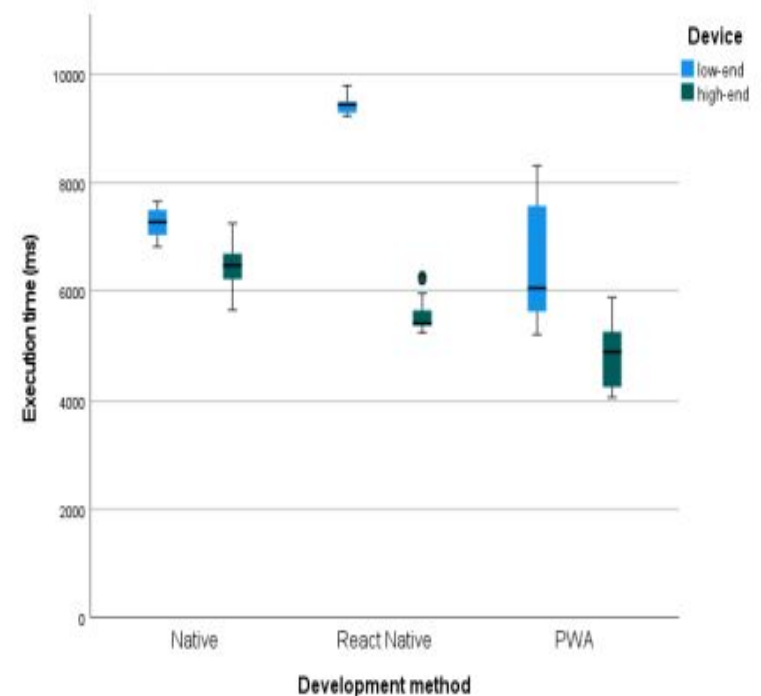
- Với thiết bị Android cấp thấp, PWA: 603 MB (s.d. 0,82 MB), native 1701 MB (s.d. 0,98 MB), React Native 1842 MB (s.d. 1,41 MB) và Flutter 1926 MB (s.d. 0,81 MB)
- Trên thiết bị Android cao cấp, PWA có tổng dữ liệu trung bình được truyền thấp nhất là 605,5 MB (s.d. 1,05 MB), native 1758 MB (s.d. 0,93 MB), React Native 1912 MB (s.d. 1,27 MB) và Flutter 2086 MB (s.d. 1,70 MB)



III. Network-Intensive Test

Thời gian thực thi trên iOS

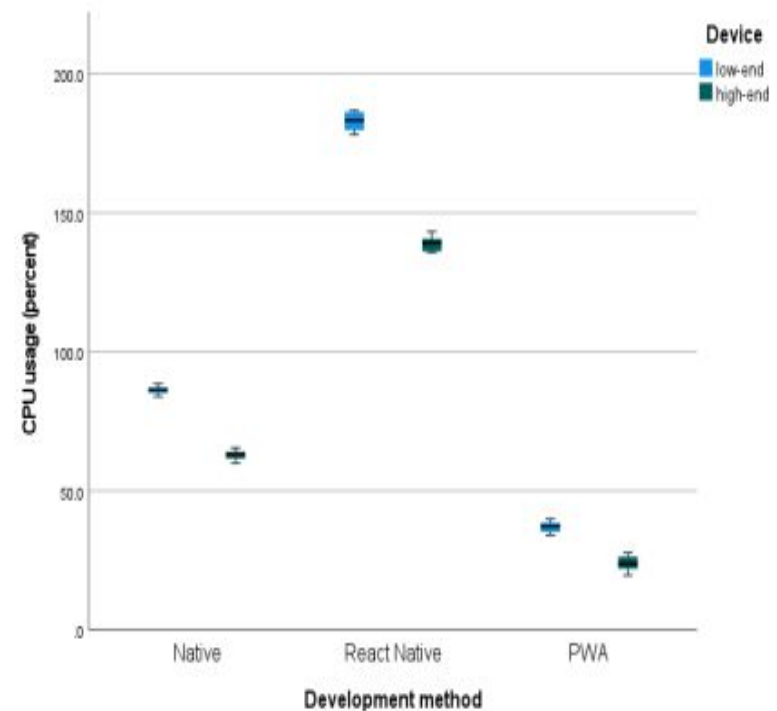
- Với thiết bị iOS cấp thấp, PWA thấp nhất 6049 ms (s.d. 1055,14 ms), native 7273 ms (s.d. 764,48 ms) và React Native 9442,5 ms (s.d. 140,40 ms)
- Trên thiết bị iOS cao cấp, PWA: 4917 ms (s.d. 606,26 ms), React Native 5438,5 ms (s.d. 280,24 ms) và native 6468,5 ms (s.d. 404,59 ms)



III. Network-Intensive Test

Mức sử dụng CPU trên iOS

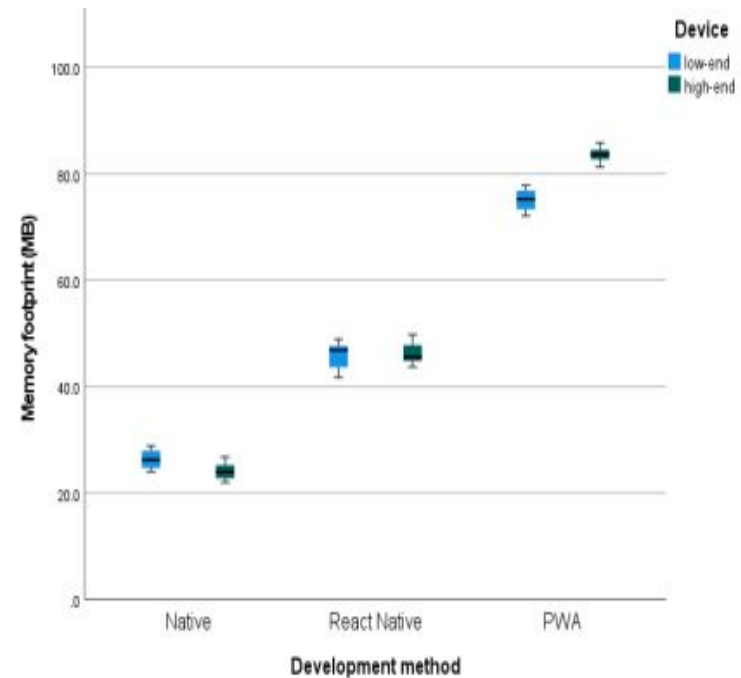
- Với thiết bị cấp thấp iOS, mức sử dụng CPU trung bình cho PWA là thấp nhất ở mức 37,2% (s.d. 1,89%), native 86,4% (s.d. 1,33%) và React Native 183,2% (s.d. 3,28%)
- Trên thiết bị cao cấp iOS, PWA có mức sử dụng CPU trung bình thấp nhất ở mức 23,8% (s.d. 2,28%), native 63,2% (s.d. 1,59%) và React Native 139,2% (s.d. 2,49%)



III. Network-Intensive Test

Mức chiếm dụng bộ nhớ trên iOS

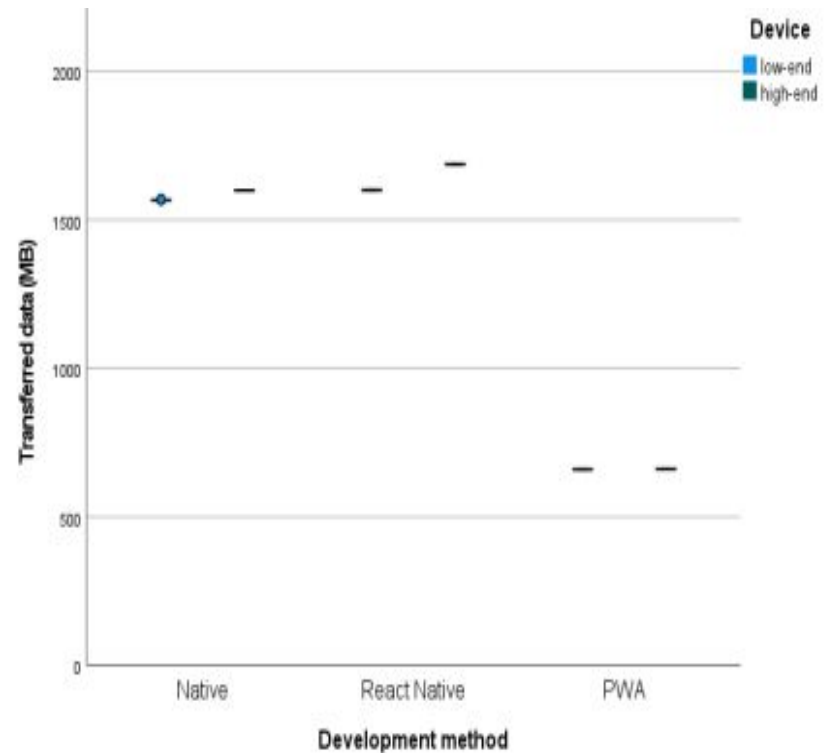
- Với thiết bị iOS cấp thấp, mức chiếm bộ nhớ trung bình cho native là thấp nhất với 26,2 MB (s.d. 1,64 MB), React Native 46,8 MB (s.d. 2,42 MB) và PWA 75,2 MB (s.d. 1,93 MB)
- Trên thiết bị cao cấp iOS, native có dung lượng bộ nhớ trung bình thấp nhất là 24 MB (s.d. 1,56 MB), React Native 45,5 MB (s.d. 1,95 MB) và PWA 83,6 MB (s.d. 1,32 MB)



III. Network-Intensive Test

Dữ liệu được truyền trên iOS

- Với thiết bị iOS cấp thấp, tổng dữ liệu được truyền cho PWA là thấp nhất với 660 MB (s.d. 0,95 MB). Native và React Native có tổng dữ liệu được truyền trung bình là 1567 MB (s.d. 1,22 MB) và 1601 MB (s.d. 1,72 MB)
- Trên thiết bị iOS cao cấp, PWA có tổng dữ liệu trung bình được truyền thấp nhất là 661,5 MB (s.d. 0,74 MB). Native và React Native có tổng dữ liệu được truyền trung bình là 1599,5 MB (s.d. 0,93 MB) và 1687,5 MB (s.d. 1,35 MB)



Kết quả chung:

- React Native có thời gian thực thi cao hơn đáng kể so với các phương pháp khác trong CPU-intensive test ở cả 2 nền tảng
- Lớp thiết bị có ảnh hưởng lớn đến thời gian thực thi của tất cả các phương pháp phát triển trong CPU-intensive test trên cả hai nền tảng, ngoại trừ native Android app
- Với các CPU-intensive test trên iOS, PWA cho thấy mức sử dụng CPU thấp hơn đáng kể so với các phương pháp khác
- Trong các Memory-intensive test trên Android, native và PWA nhanh hơn đáng kể về thời gian thực thi và sử dụng ít bộ nhớ hơn so với React Native và Flutter
- PWA không thể hoàn thành Memory-intensive test trên thiết bị iOS cấp thấp.
- Trên iOS, native app có thời gian thực thi cao nhất trong memory-intensive test

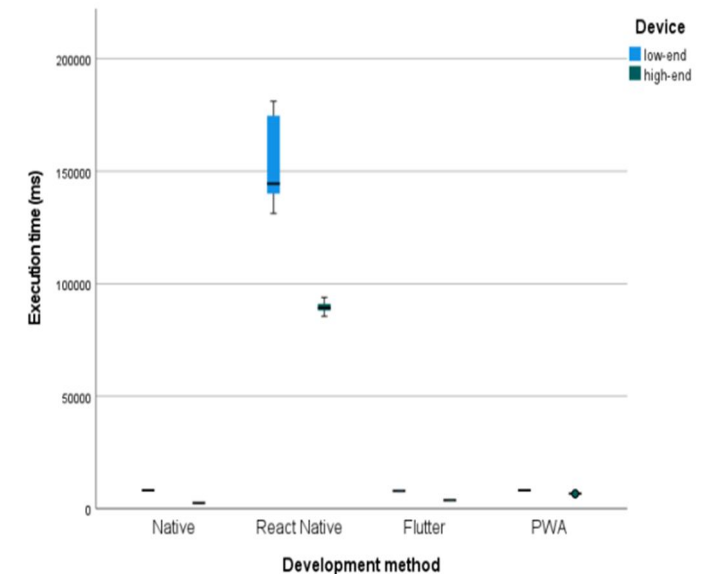
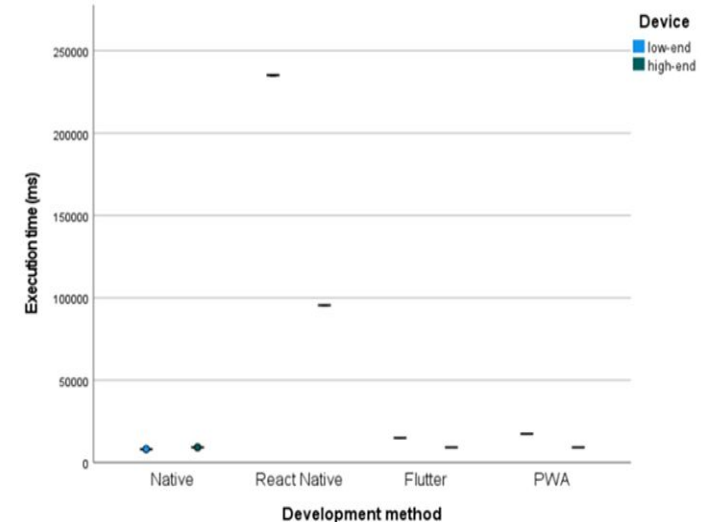
Kết quả chung:

- Mức độ sử dụng CPU cao đã được quan sát thấy trong các kết quả của memory-intensive test
- Lớp thiết bị có ảnh hưởng đáng kể đến thời gian thực thi các memory-intensive test
- Trên Android, Flutter có thời gian thực thi nhanh nhất trong các network-intensive test
- Trong các network-intensive test, Flutter không ổn định trên nền tảng Android và không thể hoàn thành test trên nền tảng iOS.
- PWA có tổng dữ liệu được truyền thấp nhất trong các network-intensive test trên cả hai nền tảng
- Lớp thiết bị có ảnh hưởng lớn đến thời gian thực thi các network-intensive test

Tổng kết

1. Tại sao React Native có thời gian thực thi cao hơn đáng kể so với các phương pháp khác trong CPU-intensive test ở cả 2 nền tảng

- Kiến trúc của React Native làm cho thời gian thực thi cao.
 - Giao tiếp giữa JavaScript và native ở React Native cần chuyển từ JSON sang đối tượng và ngược lại.
- Tốc độ chạy của JavaScript code phụ thuộc vào công cụ JavaScript mà nó chạy.
 - React Native không sử dụng JIT nên khiến cho tốc độ thực hiện chậm hơn



2. Lớp thiết bị có ảnh hưởng lớn đến thời gian thực thi của tất cả các phương pháp phát triển trong CPU-intensive test trên cả hai nền tảng, ngoại trừ native Android app:

- Yếu tố đầu tiên là về sức mạnh của phần cứng
 - So với thiết bị cấp thấp, thiết bị Android cao cấp có CPU mạnh hơn với tốc độ xung nhịp cao hơn (2840 MHz so với 2450 MHz) và băng thông bộ nhớ cao hơn (34,1 GB / s so với 29,8 GB / s)
 - Tương tự, so với thiết bị cấp thấp, thiết bị iOS cao cấp được sử dụng bởi một CPU mạnh hơn, tốc độ xung nhịp cao hơn (3100 MHz so với 2380 MHz) và băng thông bộ nhớ cao hơn (42,7 Gbit / s so với 14,9 Gbit / s)
- Yếu tố thứ hai nằm ở tuổi của phần mềm
 - Thiết bị Android cấp thấp hỗ trợ phiên bản cũ hơn của Android 9, trong khi thiết bị cao cấp hỗ trợ Android 11

3. Với các CPU-intensive test trên iOS, PWA cho thấy mức sử dụng CPU thấp hơn đáng kể so với các phương pháp khác:

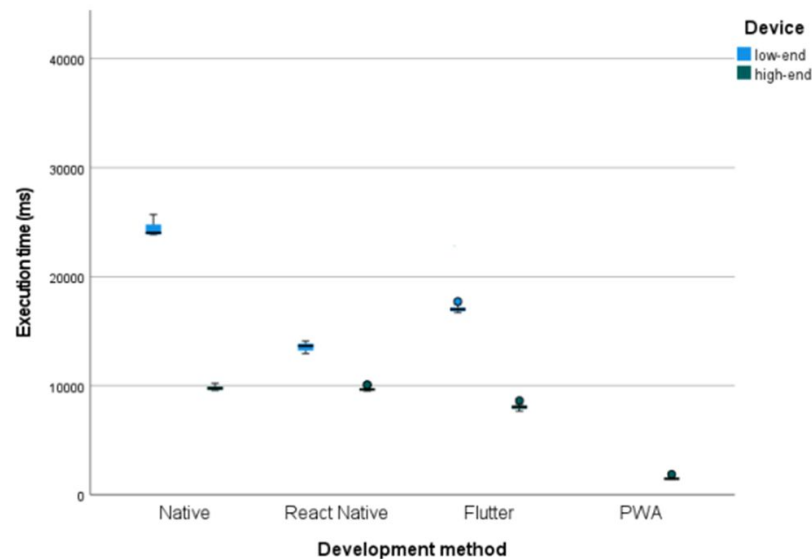
- PWA trên iOS chạy trong trình duyệt và trình duyệt đó (Safari) quản lý việc sử dụng tài nguyên của ứng dụng, bao gồm việc sử dụng CPU
- Hạn chế sử dụng tài nguyên do Apple áp đặt trên các ứng dụng trình duyệt

4. Trong các Memory-intensive test trên Android, native và PWA nhanh hơn đáng kể về thời gian thực thi và sử dụng ít bộ nhớ hơn so với React Native và Flutter:

- Native và PWA được thiết kế để chạy trên các nền tảng tương ứng, hệ điều hành Android cho native và trình duyệt cho PWA
- Việc hỗ trợ nhiều nền tảng trong React Native và Flutter đã ảnh hưởng đến hiệu quả sử dụng tài nguyên và hiệu suất của chúng trong các memory-intensive test trên Android.

5. PWA không thể hoàn thành Memory-intensive test trên thiết bị iOS cấp thấp:

- Mức chiếm dụng bộ nhớ lớn trên các thiết bị iOS cấp thấp
- Thiết bị iOS cao cấp hoàn thành test vì nó được trang bị dung lượng bộ nhớ khả dụng-4 GB so với 2 GB ở thiết bị cấp thấp

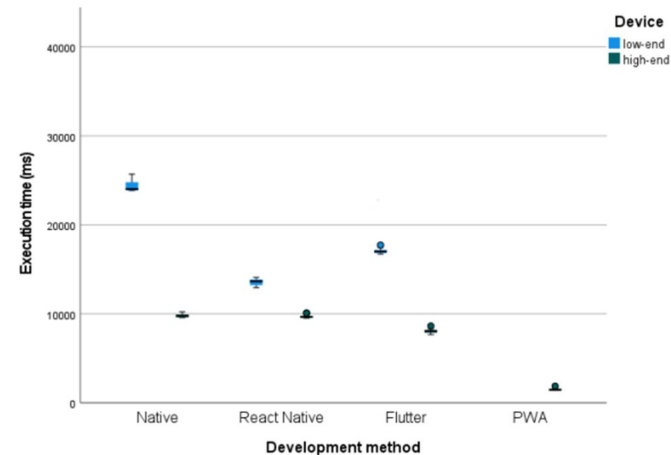
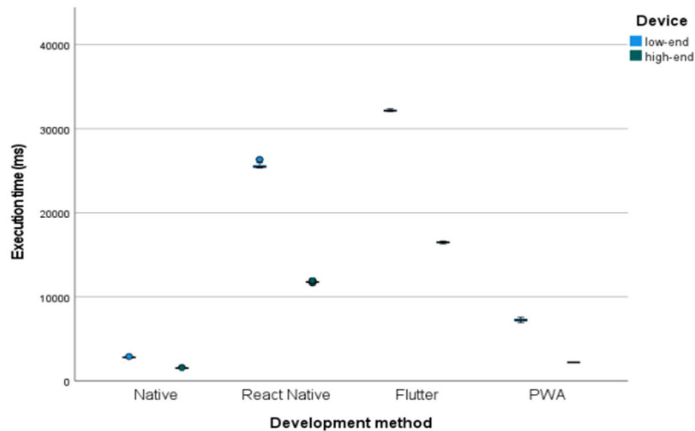


6. Mức độ sử dụng CPU cao trong các kết quả của memory-intensive test:

- Một đơn vị bộ nhớ gồm các thành phần nhỏ hơn gọi là page. Hệ thống quản lý bộ nhớ cấp phát bộ nhớ cho một process bằng cách chọn ra các trang tự do, việc chọn này sử dụng nhiều tài nguyên
- Hệ thống quản lý bộ nhớ thực hiện sao chép một số các trang bộ nhớ vào đĩa để giải phóng bộ nhớ, quá trình này yêu cầu khả năng xử lý, dẫn đến việc sử dụng CPU cao hơn trong các tác vụ sử dụng nhiều bộ nhớ

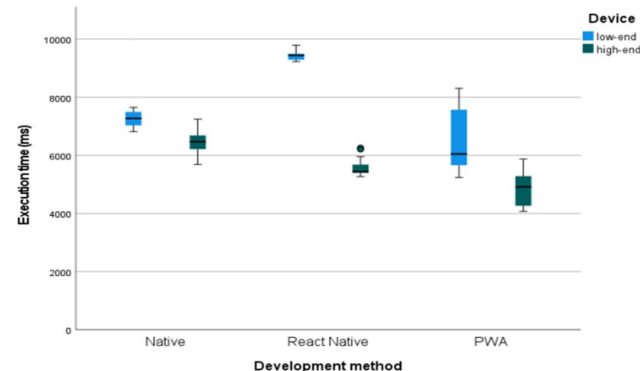
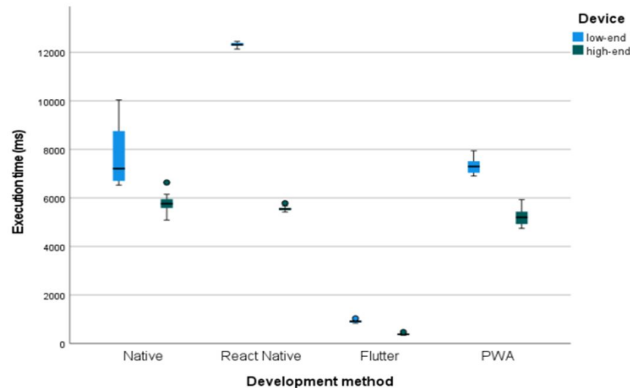
7. Lớp thiết bị có ảnh hưởng đáng kể đến thời gian thực thi các memory-intensive test:

- Các thiết bị cao cấp trên cả Android và iOS đều có CPU mạnh hơn. Ngoài ra, chúng còn được trang bị và các đơn vị bộ nhớ lớn hơn và nhanh hơn, bộ nhớ lớn gấp đôi
- Nhiều phần mềm được cập nhật hơn trên các thiết bị cao cấp



8. Trong các network-intensive test, Flutter có thời gian thực thi nhanh nhất nhưng không ổn định trên Android và không thể hoàn thành test trên iOS:

- Lỗi do quá nhiều kết nối mạng vẫn mở khi nó đang gọi API.
- Thực hiện các request HTTP qua nhiều kết nối cũng là lý do đằng sau ứng dụng Flutter nhanh hơn thời gian thực thi trong các network-intensive test

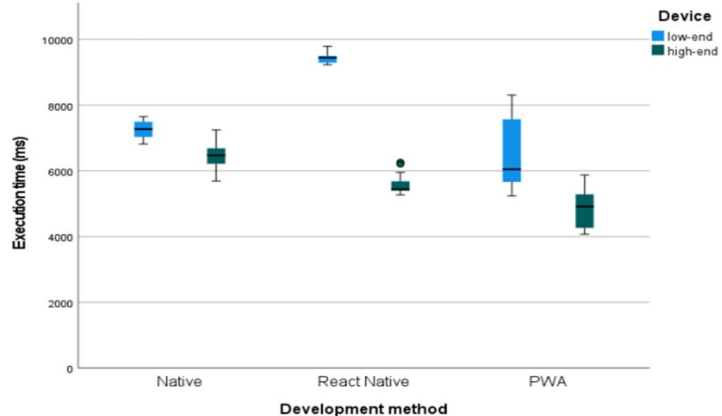
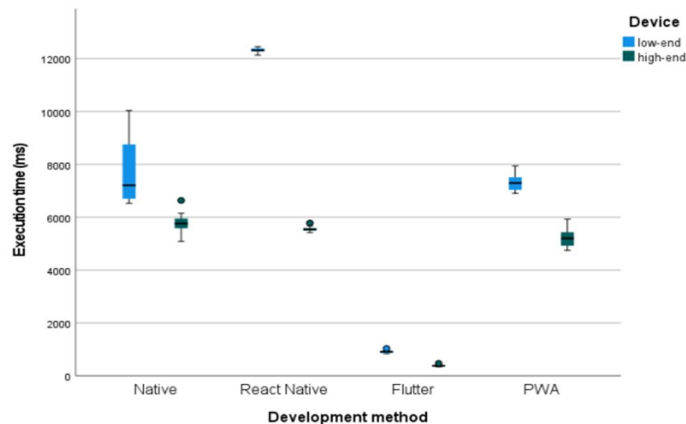


9. PWA có tổng dữ liệu được truyền thấp nhất trong các network-intensive test trên cả hai nền tảng

- Yếu tố góp phần ảnh hưởng đến kích thước của dữ liệu được truyền qua mạng là thuật toán được sử dụng trong nén HTTP
- PWA sử dụng thuật toán nén “br” là thuật toán nén hiện đại nên dữ liệu được nén tốt hơn
- Các trình duyệt tối ưu hóa việc nén so với thuật toán nén “gzip” của các phương pháp phát triển còn lại

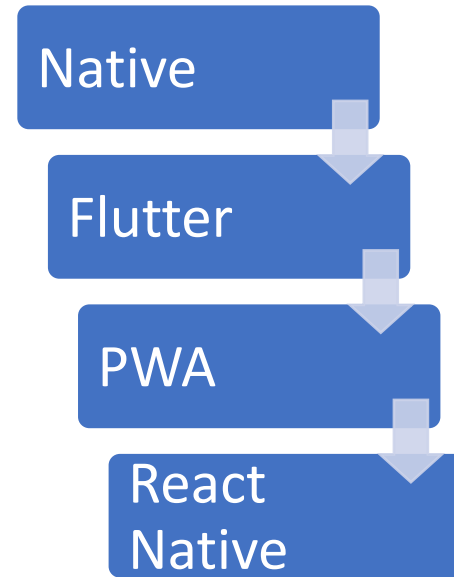
10. Lớp thiết bị có ảnh hưởng lớn đến thời gian thực thi các network-intensive test:

- Do mức độ sử dụng CPU cao, Nhiều các lệnh gọi đồng thời tới API back-end và serialize/de-serialize dữ liệu là một trong những tác vụ đòi hỏi nhiều process nhất
- Các thiết bị cao cấp trên cả hai nền tảng đều có phần cứng tốt hơn, CPU nhanh và hiệu quả hơn, các phiên bản hệ điều hành mới hơn



Kết quả chung cho từng phương pháp phát triển:

- Xếp hạng kết quả tổng thể:



- Native apps có tổng thể tốt nhất về hiệu suất và sử dụng tài nguyên tương ứng trên nền tảng Android và iOS.
- Flutter và PWA xếp thứ 2 và thứ 3. Tệ nhất là React Native

Kết quả chung cho so với các nghiên cứu trước:

- Nghiên cứu của Corral et al đã so sánh CNDNT với native Android trong một số test sử dụng nhiều tài nguyên, tập trung chủ yếu vào thời gian thực thi
 - Nghiên cứu của Dalmaso et al đánh giá bộ nhớ, CPU và mức tiêu thụ điện của hai CNDNT trên Android. Kết quả của họ cho thấy PhoneGap, một CNDNT dựa trên web, có kết quả tổng thể tốt hơn, đặc biệt ở network-intensive
 - Nghiên cứu tương tự của Willocx et al, đo mức sử dụng tài nguyên của một số CNDNT chỉ ra rằng mức sử dụng CPU cao hơn và mức chiếm dụng bộ nhớ lớn hơn của CNDNT so với native app
 - Trong nghiên cứu gần đây, Biørn-Hansen et al. đã đánh giá việc sử dụng tài nguyên và chi phí hiệu suất của một số CNDNT so với bản phát triển gốc trên Android có kết quả: hiệu suất thấp hơn và mức sử dụng tài nguyên cao hơn của CNDNT so với native
- > Các phương pháp native vẫn có ưu thế hơn về hiệu suất và sử dụng tài nguyên nhưng CNDNT hiện đại đang bắt kịp và trong một số trường hợp hoạt động tốt hơn các native apps.

Trải nghiệm của nhà phát triển



I. Tiêu chí và chỉ số đánh giá

Số dòng code:

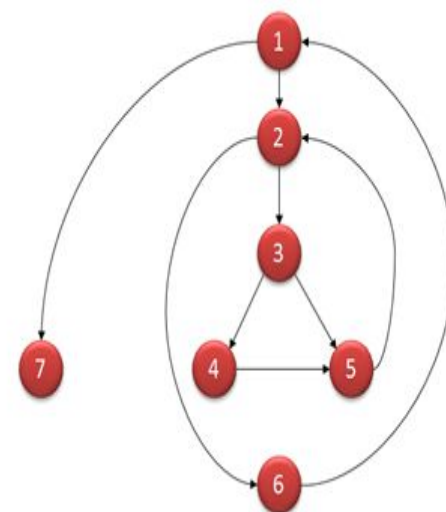
- Dòng mã (LOC) ước tính nỗ lực của việc viết một chương trình cũng như khả năng bảo trì khi một chương trình được viết
- Các loại LOC phổ biến:
 - SLOC (Source LOC) thường đề cập đến số lượng dòng mã nguồn không bao gồm nhận xét và dòng trống
 - LLOC (Logic LOC) đo số lượng các câu lệnh có thể thực thi



I. Tiêu chí và chỉ số đánh giá

Độ phức tạp Cyclomatic:

- Độ phức tạp Cyclomatic là một số liệu phần mềm định lượng để đo độ phức tạp của mã
- Về mặt toán học, độ phức tạp Cyclomatic của một chương trình được tính toán dựa trên đồ thị luồng điều khiển của nó
- Độ phức tạp M sau đó được định nghĩa là $M = E - N + 2P$, trong đó E là số cạnh của đồ thị, N là số nút của đồ thị và P là số các thành phần được kết nối



I. Tiêu chí và chỉ số đánh giá

Thời gian build:

- “Build” trong phát triển phần mềm được định nghĩa là quá trình chuyển đổi mã nguồn thành một dạng có thể được chạy trên máy tính
- Đây là một quá trình gồm nhiều bước bao gồm biên dịch, trans-compilation và gói mã nguồn và các tệp tài nguyên vào một biểu mẫu thực thi.



II. Công cụ và phương pháp

Số dòng code:

- Đầu tiên, chỉ định loại tệp nào cần sử dụng
- Sau khi chỉ định các tệp, công cụ LOC tùy chỉnh giúp tính tổng dòng mã nguồn trong các tệp đã được chỉ định sau khi loại trừ:
 - Dòng trống
 - Chú thích
 - Bất kỳ việc đóng hoặc mở dấu ngoặc nhọn mà không có bất kỳ mã logic nào trong dòng đó
 - từ khóa return
 - Bất kỳ dòng nào có mô-đun / lớp nhập và xuất
 - Các dòng chỉ có chú thích được tạo tự động
 - các thẻ trong mã liên quan đến hỗ trợ người dùng đọc mã



II. Công cụ và phương pháp

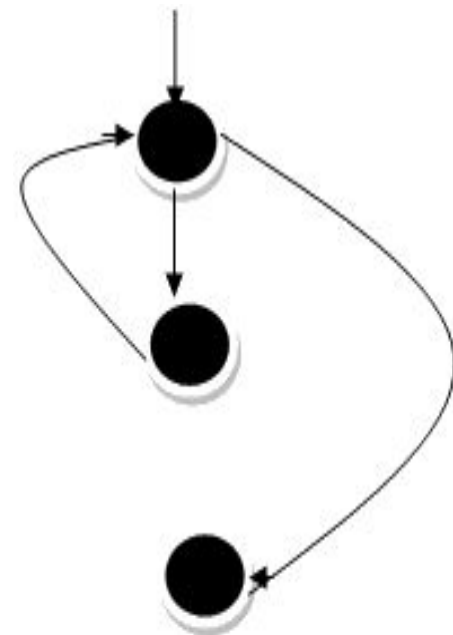
Độ phức tạp chu kì:

- Độ phức tạp Cyclomatic được đo bằng cách sử dụng một số công cụ phân tích mã hiện có.

Native Android, PWA và React Native được đo lường bởi SonarQube

- Ứng dụng Flutter được đo lường bằng cách sử dụng Dart Code Metrics

- Ứng dụng iOS gốc dựa trên Swift được đo bằng Lizard, một công cụ phân tích mã hỗ trợ nhiều ngôn ngữ lập trình



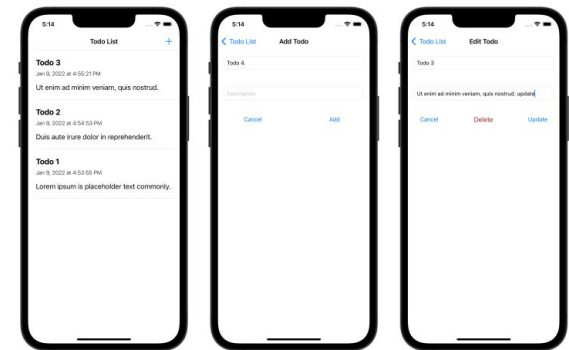
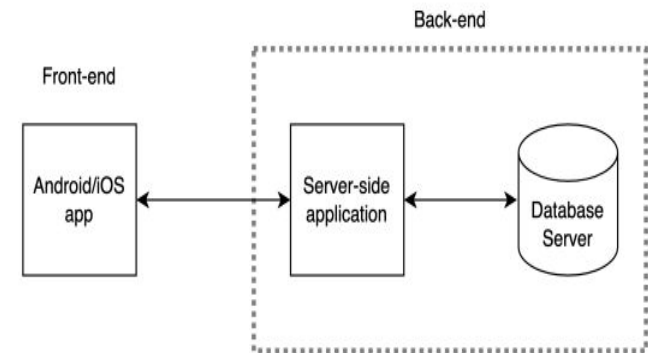
II. Công cụ và phương pháp

Thời gian build:

- Các phép đo thời gian build được thực hiện trên ba môi trường thời gian chạy khác nhau: Android, iOS và trình duyệt.
- Đối với phiên bản Android của ứng dụng - bao gồm cả React Native, Flutter và native – đo thời gian build trong Android Studio (2020.3.1 Patch 2). Các ứng dụng được build bằng lệnh Make Build, thời gian build đã được ghi vào “Event Log”.
- Trên iOS, Xcode (13.0) để đo thời gian build, bao gồm cả React Native, Flutter và native. Hiển thị thời gian build được kích hoạt bằng cách gõ lệnh trong Mac Terminal (defaults write com.apple.dt.Xcode ShowBuildOperationDuration YES)

III. Thiết kế và triển khai ứng dụng

- Ứng dụng được thiết kế cho phần này là ứng dụng cho phép người dùng tạo, cập nhật và xóa các mục việc cần làm cũng như xem tất cả các mục cần làm trong danh sách
- Ứng dụng kết nối với phía máy chủ qua RESTful API
- Kiến trúc của hệ thống này dựa trên thiết kế ứng dụng ba tầng: giao diện người dùng (front-end), ứng dụng – nơi xử lý dữ liệu và tầng dữ liệu (back-end).



IV. Tổng kết

Số dòng code (LOC)

- Ứng dụng React Native có LOC nhỏ nhất là 174, PWA với LOC là 185. Flutter có LOC là 224.
- Số LOC của các native là lớn hơn đáng kể. Cụ thể, iOS native có 352 LOC và ứng dụng Android native có LOC lớn nhất là 371.

Development Method	LOC
Native Android	371
Native iOS	352
React Native	174
Flutter	224
PWA	185

IV. Tổng kết

Độ phức tạp Cyclomatic

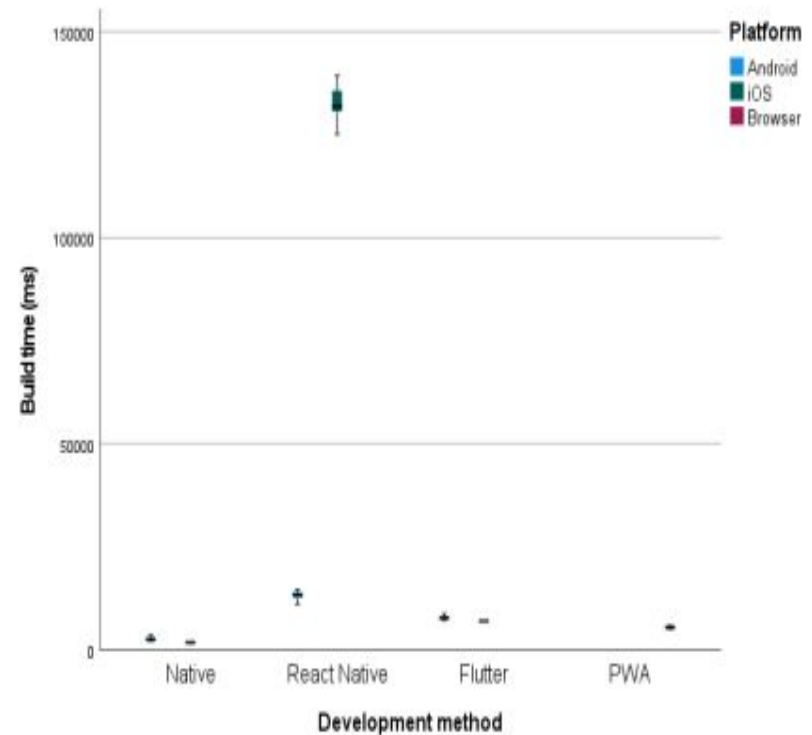
- PWA có độ phức tạp Cyclomatic là 39. React Native: 40. Flutter là 42
- Độ phức tạp theo Cyclomatic của native iOS và native Android: 53 và 63.

Development Method	Cyclomatic Complexity
Native Android	63
Native iOS	53
React Native	40
Flutter	42
PWA	39

IV. Tổng kết

Thời gian build

- Trên Android, native: 2604 ms (s.d. 450 ms). Flutter: 7789 ms (s.d. 490 ms). React Native: 13490 ms (s.d. là 1078 ms).
- Trên iOS, native: 1848 ms (s.d. là 168 ms), Flutter với 7091 ms (s.d. là 196 ms), React Native với 131897 ms (s.d. 3454 ms)
- PWA chỉ build trên trình duyệt và có thể chạy trên bất kỳ nền tảng nào có trình duyệt. PWA là 5494 ms (s.d. 307 ms).



V. Phân tích

❖ Kết quả chung:

- React Native, PWA và Flutter có LOC thấp hơn đáng kể so với native Android và native iOS.
- Các Ứng dụng PWA và React Native có LOC và độ phức tạp Cyclomatic thấp nhất.
- Độ phức tạp Cyclomatic của mã React Native, PWA và Flutter thấp hơn đáng kể so với native iOS và native Android.
- Native iOS và native Android có thời gian build nhanh nhất.
- Thời gian build của PWA nhanh hơn so với Flutter và React Native, nhưng chậm hơn so với native Android và native iOS.

V. Phân tích

Development Method	LOC
Native Android	371
Native iOS	352
React Native	174
Flutter	224
PWA	185

1. React Native, PWA và Flutter có LOC thấp hơn đáng kể so với native Android và native iOS

Do chúng là mẫu hình Declarative so với Imperative của các native

```
getRemoteData("example.com")  
  .then(parseData)  
  .then(handleParsedData)  
  .onError(displayError)
```

```
getRemoteData("example.com", { data, error in  
  if error == nil {  
    parseData(data, { parsed, error in  
      if error == nil {  
        handleParsedData(parsed, { error in  
          if error != nil {  
            displayError(error)  
          }  
        })  
      } else {  
        displayError(error)  
      }  
    })  
  } else {  
    displayError(error)  
  }  
}
```

V. Phân tích

2. Các ứng dụng PWA và React Native có LOC và độ phức tạp Cyclomatic thấp nhất:

- Do cách các ứng dụng này xử lý các lệnh gọi API và tuần tự hóa/giải mã dữ liệu (từ JSON)
- JavaScript là công cụ duy nhất hỗ trợ tích hợp cho JSON mà không cần phương thức hay constructor trợ giúp
- JavaScript có cú pháp tương đối đơn giản khi xử lý API gọi đến back-end trên máy chủ

Development Method	LOC
Native Android	371
Native iOS	352
React Native	174
Flutter	224
PWA	185

Development Method	Cyclomatic Complexity
Native Android	63
Native iOS	53
React Native	40
Flutter	42
PWA	39

V. Phân tích

3. Độ phức tạp Cyclomatic của mã React Native, PWA và Flutter thấp hơn đáng kể so với native iOS và native Android:

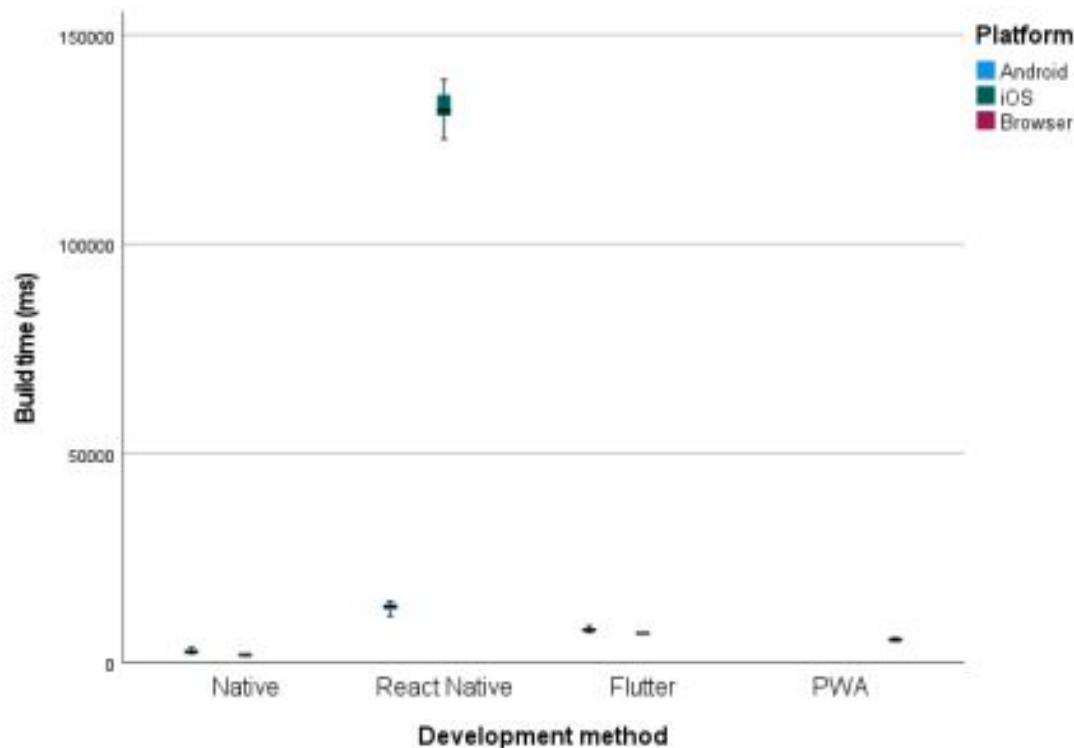
- Do chúng là mẫu hình Declarative so với Imperative của các code native

Development Method	Cyclomatic Complexity
Native Android	63
Native iOS	53
React Native	40
Flutter	42
PWA	39

V. Phân tích

4. Native iOS và native Android có thời gian build nhanh nhất

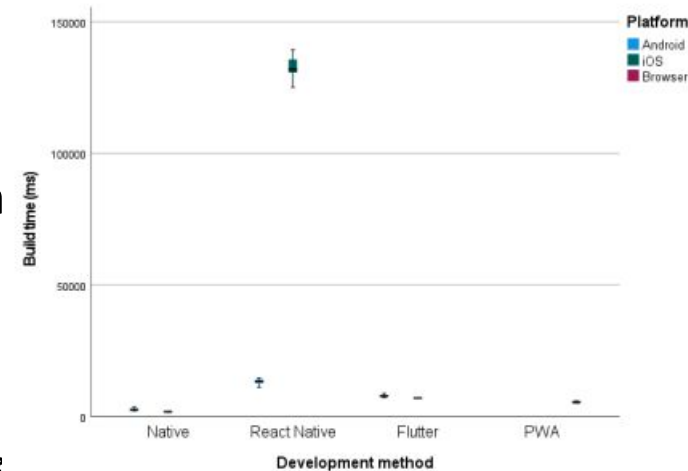
- Native app có ít hoặc không có dependencies ngoài,
- Native app chỉ được build cho một nền tảng và không có mã và tài nguyên để hỗ trợ nhiều nền tảng
- Native app có thể dùng trực tiếp luôn các thư viện của hệ điều hành



V. Phân tích

5. Thời gian build của PWA nhanh hơn so với Flutter và React Native, nhưng chậm hơn so với native Android và native iOS:

- PWA app dựa trên React và sử dụng cú pháp dành riêng cho React (JSX), nó cần thêm các công cụ để build và biên dịch ứng dụng cho trình duyệt.
- Vì vậy thời gian build bị kéo dài so với native Android và native iOS
- Khi so sánh với Flutter và React Native, PWA chỉ build cho một nền tảng, loại bỏ được chi phí cho đa nền tảng, làm cho thời gian build nhanh hơn



Hạn chế: chưa tính đến nhóm bốn chức

Bốn chức năng Scrolling, Opening Webview, Rendering ListView, Filtering khá thường gặp trong các app

Cluster	Included features
c1	chat, share content, update status, call, group chat, invite friend, video call
c2	search flight, load estimation, weather forecast,filter templates, editors, frames and shapes, emoji picker, color picker, picture templates, story templates
c3	create ads, edite ads, insight ads, campaign ads, campaign stats, ad performance, update bids, notifications, alerts
c4	e-commerce, search product, checkout product, buy product, sell product
c5	send message, read message, label message, delete message, notification, calendar integration, instant chat, video call,voice call
c6	record audio, play audio, share, upload, follow, react to uploads, rate, recommend,discover, guide
c7	connect people, meet-up, share photo/video/memory, tag person/location, react to content, watch, buy, sell, follow, notification, create events, invite guests, update status, chat, emoji picker, decorate photo/video
c8	artificial intelligence, self structuring, self reflection, mental health, reports and charts, exercises,tracker

Mục lục

1. Giới thiệu
2. Các công cụ và phương pháp
3. Thời gian khởi chạy
4. Tài nguyên sử dụng
5. Trải nghiệm của đội phát triển
6. **Kết luận**

6. Kết luận

- Một trong những lợi thế chính của React Native là cộng đồng nhà phát triển mạnh mẽ, cung cấp nhiều giải pháp thay thế cho các trường hợp sử dụng khác nhau.
- Vì React Native sử dụng JavaScript nên nó dễ tiếp cận hơn với những nhà phát triển web.
- Mặt trái là việc React Native sử dụng các cầu nối JavaScript để giao tiếp với các native module có ảnh hưởng đến hiệu suất.

6. Kết luận

- Với các công ty có đủ ngân sách và nguồn lực, Flutter là một lựa chọn tuyệt vời.
- Những người tạo Flutter nói rằng họ lấy cảm hứng từ React Native
 - việc học Flutter có thể mất nhiều thời gian hơn đối với những nhà phát triển không quen với ngôn ngữ lập trình Java/Dart.



HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.