Impact of cross platform mobile frameworks on end user performance. Flutter vs .NET 6

ROBERT-MIHAI CIUREA
POLITEHNICA Bucureşti National University for Science and Technology
Facultatea de Automatica si Calculatoare
Specializarea: Calculatoare si Tehnologia Informatiei
Email: robert_mihai.ciurea@stud.acs.upb.ro

Cristian Contasel
POLITEHNICA Bucureşti National University for Science and Technology
Facultatea de Automatica si Calculatoare
Email: cristian@hanzu.ro

Abstract
The benchmark study examines the performance of Flutter and .NET 6 technologies in mobile application development, focusing on their impact on user experience. Apps for managing lost and found animals were created and tested on Android and iOS platforms, providing relevant data for evaluation. The goal is to provide a comparison between cross-platform development and development with separate code for each platform.
Keywords: *Mobile development, Android & iOS platforms, cross-platform technology, performance evaluation*

Introduction

With the appearance of the first smartphones, the foundation was laid for the identification and development of a new field in technology, that of mobile application development. Unlike pre-installed applications in traditional mobile phones, which offered limited functionality strictly necessary to establish a basic communication between two or more users, complex systems and applications are now being discussed. They maximize the capabilities of the devices by offering an extensive range of functionalities and facilities designed to meet the full spectrum of end-user needs and preferences.

As the number of users has increased, as shown in Figure 1, so have the difficulties encountered by software developers in meeting their needs, which have become increasingly sophisticated. Mobile apps developed these days incorporate more advanced functionality, which turns development time and related costs into essential elements of the production process. At the same time, these applications ensure optimal operation, regardless of the platform (Android and/or iOS) or the technological architecture used by the diversity of mobile devices on the market. These challenges are highlighted by Dongliang You and Minjie Hu in their paper *"A Comparative Study of Cross-platform Mobile Application Development"* [1].

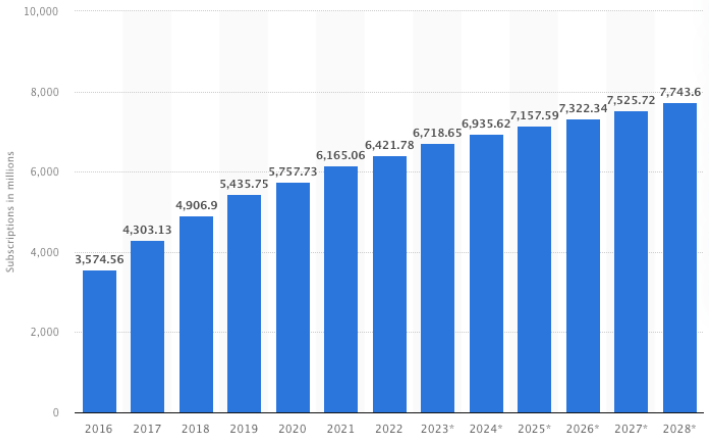

Figure 1. Mobile devices usage from 2016 to 2022 with prediction to 2028

In a comparison between a mobile device and a computer, phones have fewer resources, whether we are talking about storage space, RAM capacity or processor power, fundamental elements that can significantly influence the performance of a mobile application if it is not properly optimized. To facilitate optimization, maximizing efficiency becomes essential, this depends largely

on the desired functionalities to be implemented within the application, but also on the technologies used, be it native programming languages, frameworks dedicated to a specific platform or technologies cross-platform, as highlighted in the paper *"Performance Evaluation of Mobile Applications"* written by Anita Andonoska and Kire Jakimoski [2].

Given that Flutter and .NET 6 are two of the most popular frameworks in mobile app development today, they are viable solutions for replacing native implementations that can become extremely complex as new functionality is added in applications. In this context, these two technologies were selected to be evaluated from the perspective of their performance in relation to the end-user experience. The paper will focus on the analysis of the main metrics, to determine which of these two technologies would be the most optimal for use by developers, depending on specific circumstances. The analysis will include performance on both platforms, Android, and iOS, to provide valuable information to help developers choose the most suitable option, tailored to the needs of the project they are developing.

Defining performance analysis criteria

The execution of one app depends on 4 major aspects:

1) a given software system under execution.
2) a given hardware combination.
3) a given context.
4) a given time. In a mobile setup, the number of execution scenarios is even larger, as related by Rui Rua and João Saraiva in "A large-scale empirical study on mobile performance: energy, run-time and memory" [3].

To properly perform a performance analysis from the users' perspective using Flutter and .NET 6, both for iOS and Android, an application for managing lost and found animals was developed through which real-time analysis of the metrics was made, the most important being:

1. Execution Time: Measures how long the application takes to complete specific operations. This is crucial for understanding the app's efficiency and impacts user perception of speed and smoothness. Directly affects user experience; faster execution times are typically synonymous with a more responsive app [2]
2. Startup Time: Assesses the duration from app launch to when it becomes interactive. This metric is essential for first impressions and can influence user retention. Critical for user engagement; quicker startup times improve satisfaction, especially for frequently used apps [5]
3. Memory Usage: Evaluates the amount of RAM utilized by the app during execution. Efficient memory usage is vital for maintaining the device's overall performance and stability. Impacts the ability of the device to multitask effectively without slowing down or crashing [2]
4. CPU Usage: Measures the percentage of CPU resources utilized by the application during its operation. This helps to determine how heavy the application is on a device's processing power. Essential for understanding how the app manages processing tasks and its impact on battery life and device heat generation [6].

Each of these metrics offers a unique perspective on the performance and user experience provided by Flutter and .NET 6, enabling a balanced and thorough evaluation. Implementing these metrics will provide insights into how well each framework supports the demands of modern mobile applications, focusing on both technical performance and user-centric outcomes.

Cross-platform mobile applications development approaches.

Before approaching the model implemented to carry out the tests that aim to provide relevant data on the metrics that determine the effectiveness of the application in various contexts of use, it is imperative to examine the technologies used during the actual development. Thus, the discussion will focus on Flutter and .NET 6, as it is necessary to present fundamental information about each technology and their specific peculiarities.

1. Flutter: It is a cross-platform UI toolkit created by Google that is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services. The goal is to enable developers to deliver high-performance apps that feel natural on different platforms, embracing differences where they exist while sharing as much code as possible [7].
2. .NET 6: It is an open-source platform for building modern and performant applications (in this document only the mobile view will be taken into consideration) for iOS and Android. This represents an abstraction layer that manages communication of shared code with underlying platform code. It runs in a managed environment that

provides conveniences such as memory allocation and garbage collection. Also enables developers to share an average of 90% of their application across platforms. This pattern allows developers to write all their business logic in a single language (or reuse existing application code) but achieve native performance, look, and feel on each platform. Applications can be written on PC or Mac and compile into native application packages, such as an .apk file on Android, or an .ipa file on iOS [8].

Analysed model.

To carry out a detailed analysis of the metrics that would provide the necessary information to evaluate the effectiveness of each framework, the "Find my pet" application was developed, a solution for the management of lost and found animals. This has been implemented on both platforms, thus allowing the collection of authentic data in the widest possible spectrum of operating environments. This approach aims to simulate the experience of an anonymous user at the first interaction with the application.

The system is intended to facilitate the process of finding lost animals. There will be 2 types of users, those who have lost their pet and those who have found an unknown animal. The architecture of the "Lost and Found" system can be structured in several components, as can be seen in Figure 2:

1. User interface: the component through which users interact with the application. It should be friendly and intuitive, giving users a nice and easy experience while using the app.
2. Animal Information Upload Module: includes a form which will allow users to upload photos and detailed descriptions about the animal with information about the location where the animal was lost or found.
3. Data Filtering Module: allows users to filter information about lost or found animals according to specific criteria such as breed, color, gender, location, etc. This module will use the information loaded into the database and return only the information relevant to the user.
4. Database management module stores all uploaded information about animals lost or found. This module includes a database management system data, which ensures quick access to information and data protection against loss or unauthorized access.

In general, the architecture of the "Lost and Found" system is structured based on a client-server model. The client component comprises the user interface along with the modules responsible for loading and filtering information, while the server segment consists of the database management module, the location module, and the notification module. This structure facilitates efficient information management, simultaneously guaranteeing optimal performance and a satisfactory user experience.

Additionally, implementing a relational database supports the process of filtering and managing information, ensuring data integrity and consistency. Using a modular architecture also makes application development and maintenance easier. This minimizes dependencies between system components and provides the flexibility to replace or enhance existing modules without disrupting the overall system operation.

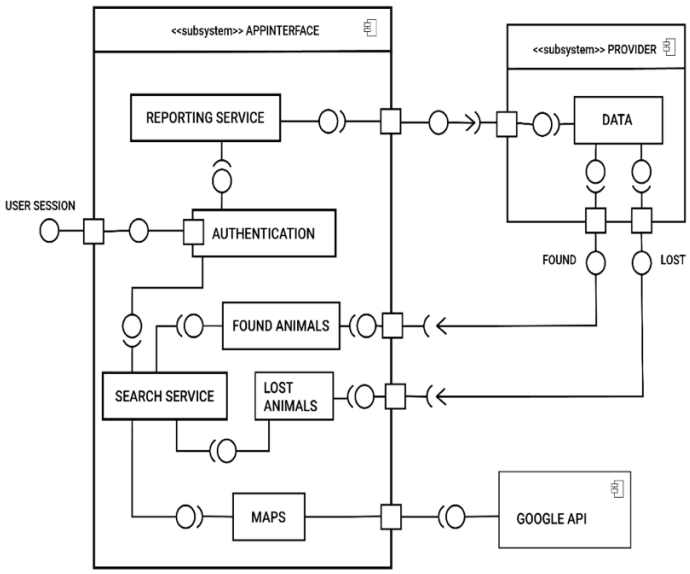

Figure 2. Decomposition into subsystems and the responsibilities of each subsystem

To conduct a detailed analysis of the performance metrics using the "Find my pet" application developed with Flutter and .NET 6, the following steps and tools/methods were approached:

1. Execution Time: Dart DevTools was used for the Flutter application to handle performance profiling and Visual Studio Diagnostic Tools for .NET 6. Benchmarks for crucial operations such as loading times, data processing, and response to user inputs were set, and these times were logged during typical usage scenarios.
2. Start-up Time: For the analysis in question, no significant differences were found at the level of the evaluation method between the two technologies studied; however, it was necessary to use specialized tools adapted to the specific operating system on which each version of the application was running. Thus, for the Android platform it was decided to use Android Studio, because it includes integrated profiling tools, while for iOS the native profiling tools available in XCode were applied. To obtain a wide range of boot time data, repeated launches of the applications were performed, both in cold state and in warm state.
3. Memory Usage: Tools such as XCode for iOS and Android Profiler, built into Android Studio, provided detailed information on the memory used by the application being tested at runtime on various devices.
4. CPU Usage: Both apps used the native CPU analysis systems built into Android Studio and XCode IDEs (Integrated development environments), respectively. Critical moments in the app were examined, such as the process of displaying the map showing lost and found animals. Also, another significant aspect analysed on both platforms and development technologies was the synchronization with the backend to retrieve all the processed data, which was then displayed on the map according to the timeline of events (animals found or lost).

Performance comparison

Device	OS	CPU	RAM
Pixel 5	Android 14 (API 34)	Qualcomm Snapdragon 765G	4GB
Pixel 6 Pro	Android 14 (API 34)	Google Tensor	6 GB
Pixel 4	Android 10 (API 29)	Qualcomm Snapdragon 855	3 GB
Pixel 3a	Android 8 (API 27)	Qualcomm Snapdragon 670	2 GB
Pixel 2	Android 6 (API 23)	Qualcomm Snapdragon 835	2GB
iPhone 15 Pro Max	iOS 17.0	A16 Bionic	6GB
iPhone 14 Pro Max	iOS 16.0	A15 Bionic	6GB
iPhone 13 Pro	iOS 15.0	A15 Bionic	6GB
iPhone 12	iOS 14.0	A14 Bionic	4GB
iPhone 11	iOS 13.0	A13 Bionic	4GB

Table 1. Devices used to test the applications.

1. Execution Time

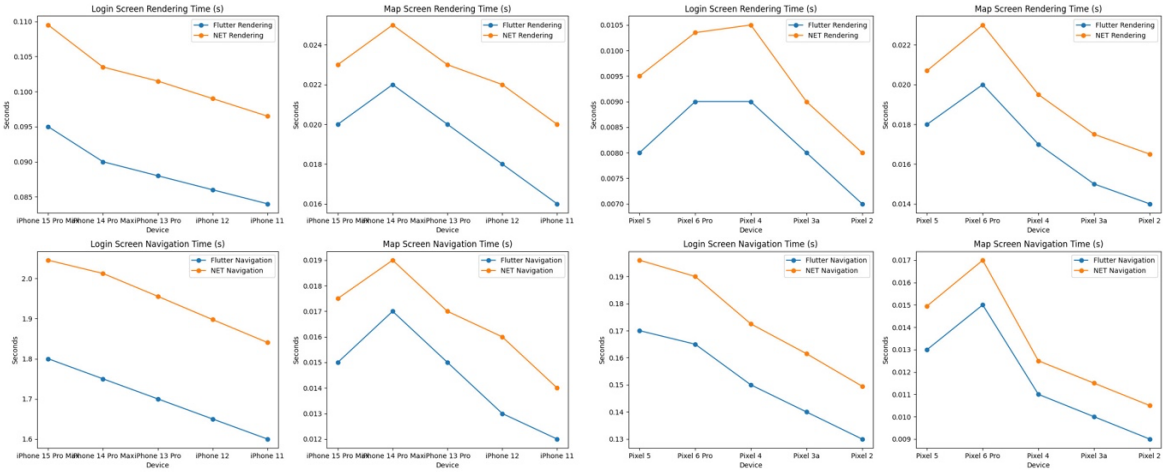

Figure 3. Rendering and Navigation time on Flutter and .NET 6

From the time perspective associated with rendering and navigation between screens, as shown in Figure 3, Flutter is found to demonstrate superior efficiency compared to .NET on both Android and iOS platforms. This superiority of Flutter can be attributed to the use of its own graphics rendering engine, Skia, for drawing widgets. In contrast, .NET 6 opts for the integration of native components specific to each platform. While this approach can provide a more "native" looking user interface, it also brings an increased reliance on the efficiency of already implemented native components. This dependency can negatively influence the response time in the process of navigation and rendering between screens, thus reducing the overall performance of the application.

2. Start-up time

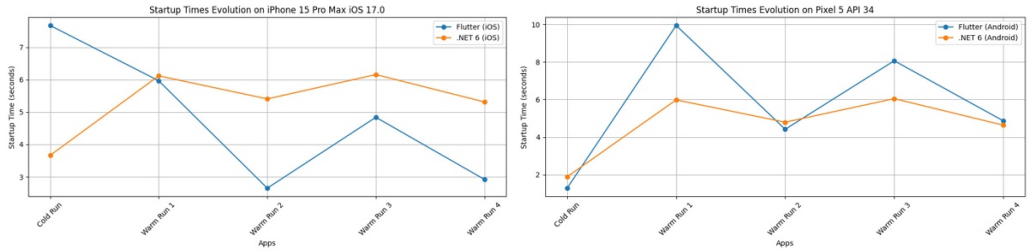

Figure 4. Start-up time on Android and iOS for Flutter and .NET 6

iPhone 15 Pro Max (iOS 17.0)

Flutter: Start-up time reduces significantly after the first load, indicating the efficiency of resource management after initialization. A slight increase observed in subsequent runs suggests sporadic background activity.

.NET 6: Demonstrates consistency in start-up time across different runs, suggesting stable resource management in iOS with minor variations.

Pixel 5 (API 34)

Flutter: Shows lower initial start-up time with noticeable variation in subsequent runs, reflecting dynamic management of system-wide optimizations.

.NET 6: Start-up time gradually increases to a maximum point and then decreases, indicating a more even approach to optimization over successive runs.

General Observations

On iOS: Analysing Figure 4, Flutter seems to perform aggressive optimizations that don't always guarantee performance consistency, while .NET 6 offers a more balanced and predictable approach.

On Android: In figure 4, both frameworks show variability in performance, but Flutter exhibits more pronounced fluctuations, possibly due to the complex interaction with the Android operating system.

3. Memory usage

Comparing the memory usage between Flutter and .NET on Android and iOS platforms, as are presented in Figure 5, it is observed that .NET experiences lower and more stable memory usage on both operating systems. On Android, this suggests that .NET is more efficient at managing memory compared to Flutter, which can be crucial in applications with strict memory usage requirements. In contrast, on iOS, .NET demonstrates better memory optimization across multiple runtimes, indicating more efficient integration with the iOS platform.

These findings indicate that while Flutter may offer advantages in rapid UI development and extended UI functionality, .NET may be preferred for applications that require minimal memory usage, especially on iOS devices. Thus, the choice between the two frameworks depends on the specific memory management needs of the application and the target platform.

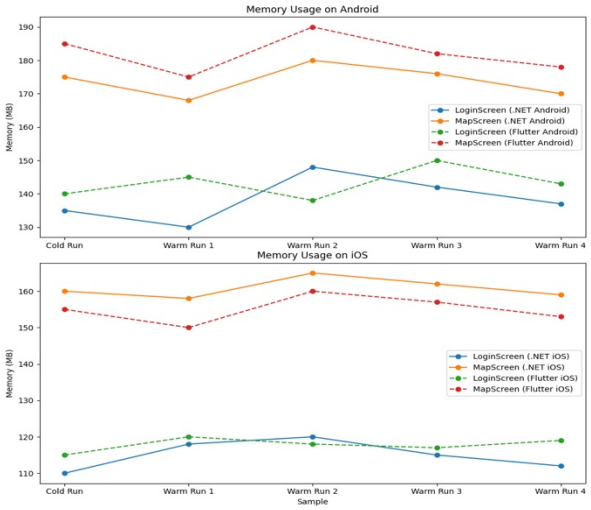

Figure 5. Memory usage on Android and iOS for Flutter and .NET 6

4. CPU usage

Android platform:

Flutter exhibits high and relatively constant CPU usage, indicating intensive computation and rendering processes.

.NET shows lower CPU usage, increasing slightly along the way, suggesting superior efficiency in resource management.

iOS platform:

Flutter starts with low CPU usage, but experiences a significant spike, possibly due to JIT compilation or intensive rendering tasks.

.NET maintains a consistently low CPU usage, indicating efficient optimization and integration with the iOS system.

General Observations:

Flutter requires more CPU resources compared to .NET, results presented in Figure 6, which can influence the choice of technology depending on the specific performance and energy efficiency requirements of the application.

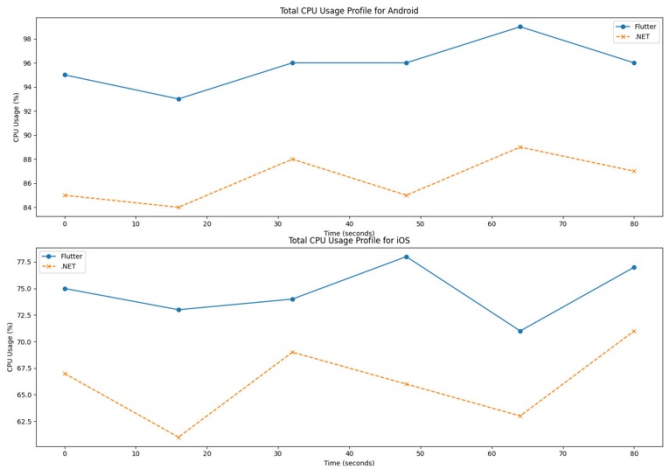

Figure 6. CPU usage on Android and iOS for Flutter and .NET 6

.NET offers more moderate CPU usage on both platforms, an important advantage for applications that require battery conservation and reduced hardware impact.

Conclusions

The study analysed the performance of two popular mobile application development frameworks, Flutter and .NET 6, focusing on their influence on end-user experience. The analysis results show notable differences between the two technologies in terms of runtime, start-up time, memory usage, and CPU consumption on both Android and iOS platforms.

Flutter has demonstrated superior efficiency in terms of running speed of applications on both platforms, possibly due to its proprietary rendering engine, Skia, which optimizes the drawing of widgets. .NET 6, using native components of the platforms, while providing a user interface that feels more "native", can suffer from dependence on the efficiency of already implemented native components.

.NET 6 showed more consistent resource management on iOS, with stable start-up times, while Flutter showed initially shorter start-up times on Android, but with noticeable variation in subsequent runs, indicating dynamic management of level optimizations of system.

Lower and more stable memory usage was also more consistent in .NET 6 framework on both platforms, suggesting more efficient integration and optimization with operating systems. This is crucial in applications where strict memory usage requirements are essential.

Flutter experienced higher and relatively constant CPU usage, which indicates compute and rendering intensive processes. On the other hand, .NET 6 showed lower CPU usage, increasing slightly along the way, suggesting superior efficiency in resource management.

How to Decide Between Flutter and .NET

Choosing between Flutter and .NET depends on several project-specific factors:

1. Performance and Resilience:
   Flutter is ideal for applications that require high graphics performance and a smooth and responsive user experience.
   .NET is preferable for applications that prioritize power efficiency, stability, and deep integration with the native platform.
2. Device Resources:
   If the application needs to run on resource-constrained devices, .NET may be more suitable due to its low memory and CPU usage.
3. Frequency of Application Use:

For frequently used applications, the fast start-up time and efficient resource management provided by .NET can improve user retention.

4. Development Complexity:

Flutter enables rapid and iterative development with a common source code for multiple platforms, ideal for projects with tight deadlines and limited budgets.

Bibliografie

[1] Dongliang You, Minjie Hu, “A Comparative Study of Cross-platform Mobile Application Development”, 12th Annual Conference of Computing and Information Technology Research and Education New Zealand, 2021.

[2] Anita Andonoska, Kire Jakimosk, “Performance Evaluation of Mobile Applications”, Conference: XIV International Conference – ETAI 2018, Republic of Macedonia, 2018

[3] Rui Rua, João Saraiva, “A large-scale empirical study on mobile performance: energy, run-time and memory”, Empirical Software Engineering (2024) 29:31, 2024

[4] Andre Luiz Nunes Martins, Cesar A. V. Duarte, Jinkyu Jeong, “Improving Application Launch Performance in Smartphones Using Recurrent Neural Network”, ICMLT '18: Proceedings of the 2018 International Conference on Machine Learning Technologies, 2018

[5] Diya Datta, Sangaralingam Kajanan, “Do app launch times impact their subsequent commercial success?”, International Journal of Big Data Intelligence 3(4):279, 2016

[6] Thomas Dorfer, Lukas Demetz, Stefan Huber, “Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features”, Procedia Computer Science Volume 175, Pages 189 – 196, 2020

[7] Aakanksha Tashildar, Nisha Shah, Rushabh Gala, Trishul Giri, Pranali Chavhan, “Application Development using Flutter”, International Research Journal of Modernization in Engineering Technology and Science Volume:02/Issue:08/August-2020 Impact Factor – 5.354, 2020

[8] Kumar Vishal, Ajay Shiram Kushwaha, “Mobile Application Development Research Based on Xamarin Platform“, 4th International Conference on Computing Sciences (ICCS), 2018