JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 6 (2018) 87-91
Wysłane: 2017-11-12
Przyjęte: 2017-11-17

Porównanie wydajności wieloplatformowego szkieletu aplikacji na platformach Android i Windows 10 Mobile.

Dawid Wieczorek*, Jakub Smołka

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule sprawdzono czy aplikacje napisane z wykorzystaniem wieloplatformowego szkieletu działają jednakowo wydajnie na wybranych systemach operacyjnych co rozwiązania natywne. Testowaniu poddano framework Xamarin.Forms porównując go z rozwiązaniami natywnymi Android SDK i Universal Windows Platform na systemach Android i Windows 10 Mobile.

Słowa kluczowe: xamarin, cross-platform, android, windows

*Autor do korespondencji.

Adres e-mail: wieczorek.dawid@outlook.com

Comparison of performance multi-platform application core on Android and Windows 10 Mobile.

Dawid Wieczorek*, Jakub Smołka

Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article examined whether applications written using the cross-platform application framework perform equally well as native solutions on selected operating systems. The Xamarin.Forms framework was tested against the native Android SDK and Universal Windows Platform frameworks for mobile systems Android and Windows 10 Mobile.

Keywords: xamarin, cross-platform, android, windows

*Corresponding author.

E-mail address: wieczorek.dawid@outlook.com

1. Wstęp

Celem artykułu jest porównanie wydajności wieloplatformowego szkieletu aplikacji napisanej z wykorzystaniem Xamarin oraz Xamarin.Forms. Analizę wykonano na dwóch platformach mobilnych – Androidzie w wersji 7 Nougat i Windows 10 Mobile. Badanie obejmuje zarówno teoretyczne aspekty wymienionego frameworku jak i testy praktyczne.

W artykule sprawdzana jest hipoteza, że tworzenie aplikacji ze wspólnym szkieletem umożliwia wydajne ich działanie niezależnie od systemu operacyjnego. Hipoteza została postawiona ze względu na to, że segment aplikacji mobilnych jest jednym z najszybciej rozwijających się segmentów aplikacji, a przed twórcami postawione jest zadanie stworzenia jednakowo wydajnych i ergonomicznych aplikacji na różne systemy mobilne takie jak Android i Windows 10 Mobile.

W klasycznym podejściu każda z aplikacji na różne systemy mobilne tworzona jest z osobna, natomiast celem twórców jest zapewnienie tej samej funkcjonalności niezależnie od systemu operacyjnego. Dla firm zajmujących się wytwarzaniem aplikacji mobilnych utrzymanie kilku zespołów deweloperskich dla oddzielnych systemów mobilnych generuje wysokie koszty implementacji nowych funkcjonalności oraz utrzymania danego produktu [1].

Wychodząc naprzeciw oczekiwaniom twórców oprogramowania powstają frameworki pozwalające na współdzielenie kodu źródłowego aplikacji między systemami operacyjnymi takie jak Xamarin. Pozwalają one na tworzenie uniwersalnych aplikacji w jak najmniejszym stopniu obciążając programistów koniecznością dopasowania do konkretnego systemu operacyjnego. Pozwala to tworzyć oprogramowanie mobilne mniejszym nakładem pracy, a co za tym idzie niższym kosztem.

2. Natywne technologie

Stworzenie wieloplatformowej aplikacji napisanej z wykorzystaniem natywnych technologii zabiera dużo czasu ze względu na konieczność przenoszenia oprogramowania między systemami. Głównym językiem, w którym stworzone są aplikacje dla systemu Windows 10 Mobile jest C#, aplikacje dla systemu Android są natomiast implementowane w języku Java. Decydując się na korzystanie z technologii natywnych powstają faktycznie dwie oddzielne aplikacje połączone wspólnymi zasobami takimi jak grafika, dźwięk. Dodając nowe funkcjonalności programiści muszą poświęcić znaczący czas na stworzenie kodu źródłowego dla obu platform. Odbija się to negatywnie na kosztach wytworzenia oprogramowania [2].

Są segmenty rynku, w których tworzenie aplikacji natywnych jest zdecydowanie lepszym rozwiązaniem niż tworzenie aplikacji wieloplatformowej. Są to między innymi aplikacje wymagające najwyższej wydajności obliczeniowej oraz gry wykorzystujące zaawansowaną grafikę 3D. Dodatkowym plusem aplikacji natywnych jest sprawniejsze działanie dotyku. Gesty takie jak szczypanie i przesuwanie działają dokładnie tak, jak było to zamierzone przez

użytkownika. Czas odpowiedzi na daną interakcję jest dokładnie taki sam jak w pozostałych systemowych i natywnych aplikacjach. Brak warstwy pośredniej zapobiega opóźnieniom dotyku [3].

3. Platforma Xamarin

3.1. Xamarin

Xamarin jest wieloplatformowym rozwiązaniem stanowiącym zunifikowane środowisko programistyczne dla deweloperów aplikacji mobilnych. Wprowadza możliwość tworzenia aplikacji w języku C# dla systemu Android oraz iOS, a także obsługuje system Windows 10 Mobile. Na systemie Android działa wykorzystując interpretowany kod IL zdolny do uruchomienia za pomocą środowiska uruchomieniowego Mono zintegrowanego ze stworzoną aplikacją. Mono działa bezpośrednio na poziomie jądra Linuxowego, nie wykorzystując maszyny Dalvik bądź środowiska uruchomieniowego ART, co pozwala na wydajne działanie aplikacji [4].

Platforma Xamarin umożliwia wykorzystywanie natywnego interfejsu SDK dla systemu Android z poziomu aplikacji. Odbywa się to poprzez wywołanie maszyny Dalvik, bądź środowiska uruchomieniowego ART za pomocą Managed Callable Wrapper, na którym uruchamiana jest natywna funkcjonalność. Rezultat wykonania przesyłany jest do środowiska uruchomieniowego Mono poprzez kanał komunikacji Android Callable Wrapper. Interakcja aplikacji napisanej w Xamarinie z maszyną Dalvik i środowiskiem ART obciążona jest sporym kosztem wydajnościowym [5, 6, 7].

3.2. Xamarin.Forms

Xamarin.Forms jest zestawem narzędzi pozwalającym w bardzo prosty sposób definiować współdzielony wygląd aplikacji. Elementy Xamarin.Forms definiowane są za pomocą Extensible Application Markup Language (XAML). To API dostarcza wiele gotowych wieloplatformowych elementów interfejsu użytkownika, takich jak gotowe strony, elementy nawigacyjne, kontenery elementów na stronach oraz kontrolki pozwalające na interakcję z użytkownikiem [8].

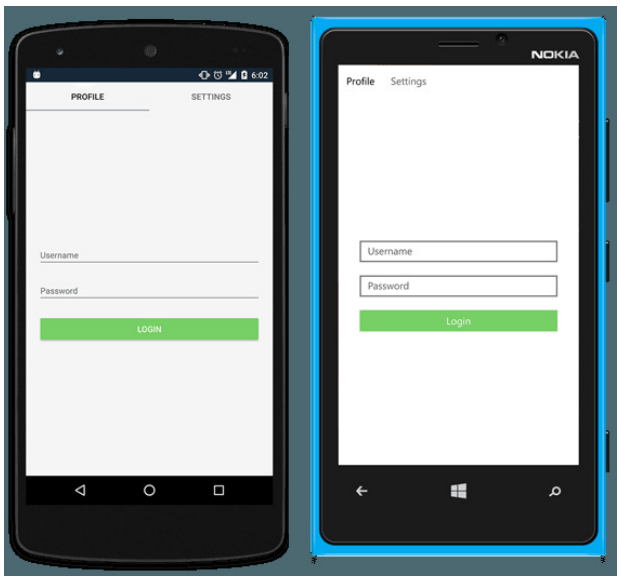
Interfejs zaprojektowany w Xamarin.Forms bez problemu może być przenoszony pomiędzy różnymi systemami operacyjnymi. Elementy graficzne zaprojektowane są w taki sposób, aby wygląd odpowiadał w jak największym stopniu systemowi operacyjnemu na którym jest uruchamiany. Jako przykład można podać wygląd elementu odpowiadającego za wprowadzanie tekstu, na systemie Android jest on wyświetlany jako podkreślone pole tekstowe, natomiast na systemie Windows 10 Mobile pole tekstowe jest z każdej strony otoczone ramką (Rys. 1)[9].

Dostosowywanie się elementów interfejsu pozwala zwiększyć atrakcyjność aplikacji, które nie odstają od aplikacji natywnych napisanych dla konkretnego systemu operacyjnego, zachowując wytyczne [10].

4. Metoda badań

W celu stworzenia testów wydajności zdecydowano się na zaprojektowanie i zaimplementowanie aplikacji testujących wykorzystując:

1) Język C# i platformę .NET
2) Język Java
3) Środowisko programistyczne Visual Studio
4) Środowisko programistyczne Android Studio
5) Emulator z systemem Windows 10 Mobile
6) Emulator z systemem Android 7



Rys. 1. Porównanie wyglądu zdefiniowanego za pomocą tego samego pliku XAML w systemie Android (po lewej) oraz Windows 10 Mobile (po prawej)

Testy zostały podzielone na kilka kategorii:

1) Test uruchamiania aplikacji
2) Testy wydajności obliczeniowej
3) Testy wydajności odczytu/zapisu danych
4) Testy prędkości połączenia internetowego

Test uruchamiania aplikacji polegał na dokładnym zmierzeniu czasu uruchamiania dla aplikacji zawierającej jeden widok od momentu naciśnięcia ikony aplikacji na pulpicie głównym, do czasu wyświetlenia widoku.

Testy wydajności obliczeniowej polegały na wykonaniu sortowania bąbelkowego tablicy zawierającej 20000 elementów oraz wykorzystaniu funkcji skrótu SHA-256 i MD-5 do haszowania 50 megabajtów danych.

Testy wydajności odczytu i zapisu danych odbywały się poprzez mierzenie czasu odczytu pliku o rozmiarze 50MB oraz 1000 plików o rozmiarze 50kB, a następnie przez zapisanie pliku o rozmiarze 100MB oraz 1000 plików o rozmiarze 50kB.

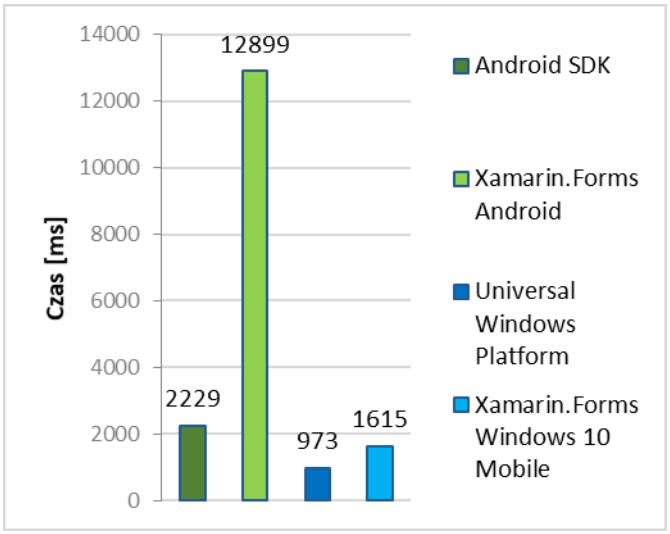
Testy zostały wykonane w aplikacjach natywnych działających na systemach operacyjnych Android i Windows 10 Mobile, oraz aplikacjach zapewniających tą samą funkcjonalność wykonanych z wykorzystaniem platformy Xamarin, które mogły być uruchomione na obydwu systemach. Pozwoliło to na stworzenie porównania wydajności aplikacji ze współdzielonym kodem do technologii natywnych na dany system operacyjny. Zestawiając różnicę między technologiami natywnymi a współdzielonym szkieletem aplikacji porównano stopień wydajności oprogramowania między wymienionymi systemami operacyjnymi [11].

5. Wyniki badań

Wyniki badań zostały przygotowane poprzez analizowanie zarejestrowanego materiału wideo w przypadku testowania czasu uruchamiania aplikacji oraz odczytanie danych zapisanych w logach aplikacji testowej. Każdy z eksperymentów został wykonany 10 razy. Obliczono następnie średni czas trwania każdego z eksperymentów.

5.1. Pomiar czasu uruchamiania aplikacji

Eksperyment ukazuje pomiar czasu uruchamiania aplikacji zawierającej ten sam interfejs użytkownika, stworzonej za pomocą rozwiązań natywnych oraz platformy Xamarin.Forms (Rys. 2). Wyniki ukazują negatywny wpływ frameworka Xamarin na wydłużenie czasu uruchomienia aplikacji. Na systemie mobilnym Android czas uruchomienia został wydłużony ponad pięciokrotnie w porównaniu do natywnego rozwiązania zaprojektowanego w Android SDK.



Rys. 2. Test - uruchamianie aplikacji

W wersji dla systemu Windows 10 Mobile wydłużenie czasu gotowości aplikacji jest sporo mniejsze, natomiast nadal znaczące - przekraczające 65%.

5.2. Testy wydajności obliczeniowej

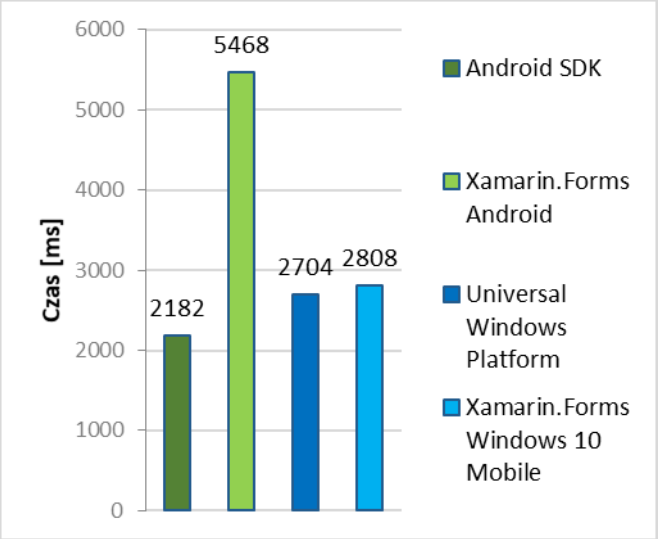
5.2.1. Sortowanie bąbelkowe

Eksperyment z przeprowadzeniem sortowania bąbelkowego 20000 elementów pokazuje znaczną różnicę między czasem wykonania sortowania w aplikacji natywnej dla systemu Android oraz w aplikacji wieloplatformowej zaimplementowanej w Xamarin.Forms (Rys. 3). Rozwiązanie przygotowane w Android SDK zakończyło sortowanie w czasie średnim o 251% krótszym od implementacji stworzonej w Xamarin.Forms.

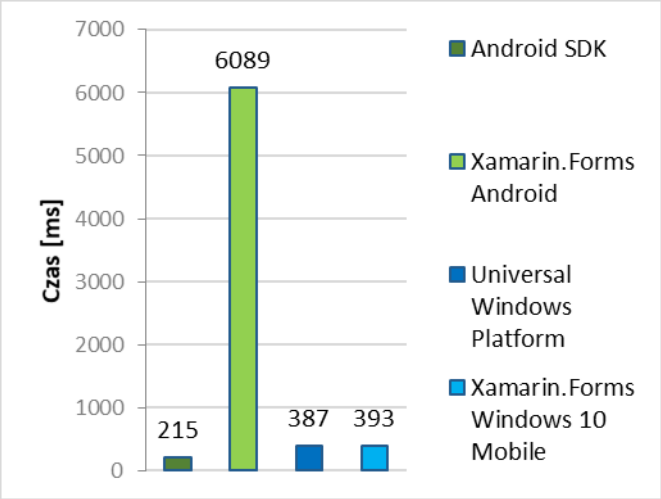
Inaczej sytuacja przedstawia się w systemie Windows 10 Mobile, gdzie sortowanie w aplikacji natywnej zakończyło się w czasie krótszym o jedyne 4%.

5.2.2. Funkcja skrótu SHA-256

Kolejny eksperyment ukazuje czas haszowania funkcją skrótu SHA-256 na poszczególnych systemach operacyjnych (Rys. 4). W tym wypadku po raz kolejny największa różnica występuje pomiędzy rozwiązaniem przygotowanym w Android SDK a Xamarin.Forms, gdzie implementacja wieloplatformowa potrzebowała na obliczenie funkcji skrótu aż 28 razy więcej czasu od aplikacji natywnej na systemie Android.



Rys. 3. Test - sortowanie bąbelkowe



Rys. 4. Test - funkcja skrótu SHA-256

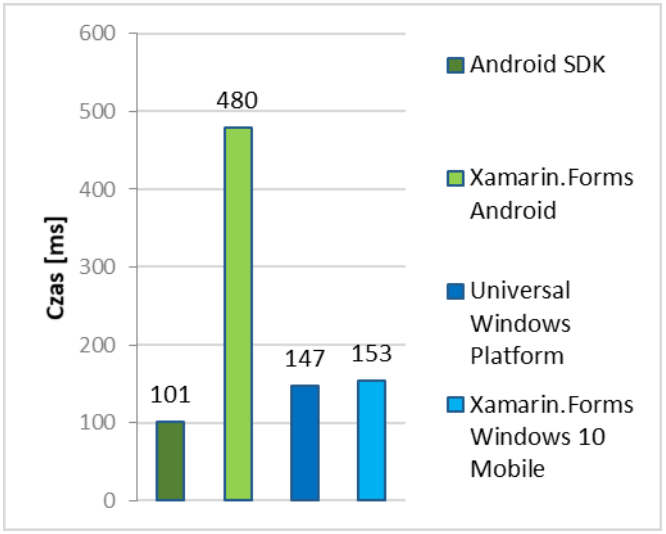
W przypadku systemu mobilnego od Microsoftu różnica między Xamarin.Forms a Universal Windows Platform jest marginalna, zarówno rozwiązanie natywne jak i wieloplatformowe zapewniają bardzo zbliżone czasy wykonania.

5.2.3. Funkcja skrótu MD5

Następny eksperyment ukazuje czas haszowania funkcją skrótu MD5 (Rys. 5). W przypadku tej funkcji skrótu różnica między implementacjami nie jest już tak duża jak

w przypadku SHA-256 na systemie Android. Xamarin.Forms wykonuje operacje w czasie o 475% dłuższym niż rozwiązanie natywne przygotowane w Android SDK.

Na systemie mobilnym od Microsoftu różnica podobnie jak w poprzednim teście sprawdzającym funkcję skrótu SHA-256 jest marginalna.

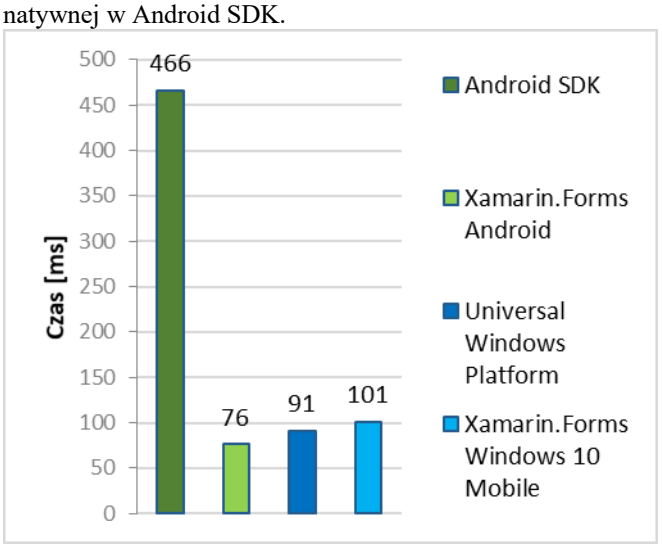


Rys. 5. Test - funkcja skrótu MD5

5.3. Testy wydajności odczytu/zapisu danych

5.3.1. Czas zapisu pliku 50MB

W tym teście został zmierzony czas zapisu pliku o rozmiarze 50MB (Rys. 6). Odwrotnie niż w poprzednich testach na systemie Android najkrótszy czas zapisu zapewnił wieloplatformowy szkielet aplikacji Xamarin.Forms. W porównaniu do natywnej technologii rozwiązanie to zapisało plik w czasie krótszym o 613% od implementacji natywnej w Android SDK.

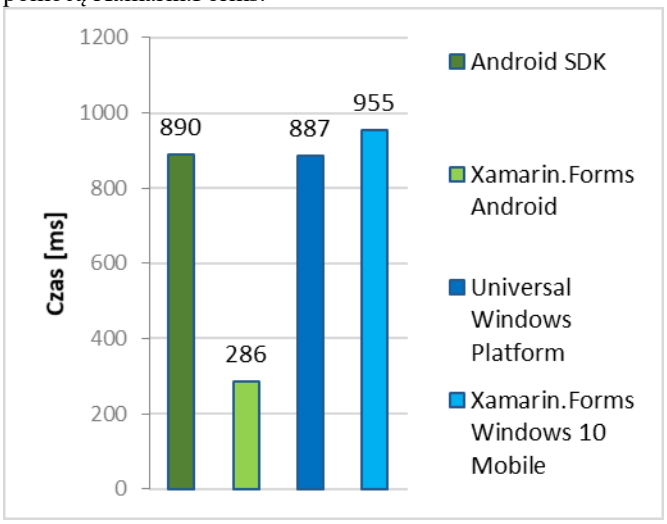


Rys. 6. Test - czas zapisu pliku 50MB

W przypadku systemu Windows 10 Mobile rozwiązanie wieloplatformowe wykonało zapis w czasie o 10% krótszym od rozwiązania przygotowanego w Universal Windows Platform.

5.3.2. Czas zapisu 1000 plików o rozmiarze 50kB

W tym eksperymencie przetestowany został zapis 1000 małych plików w pętli (Rys. 7). Po raz kolejny na systemie mobilnym od Google wydajność rozwiązania wieloplatformowego jest większa od wydajności natywnej aplikacji przygotowanej w Android SDK. W tym wypadku czas zapisu jest trzykrotnie krótszy w teście napisanym za pomocą Xamarin.Forms.

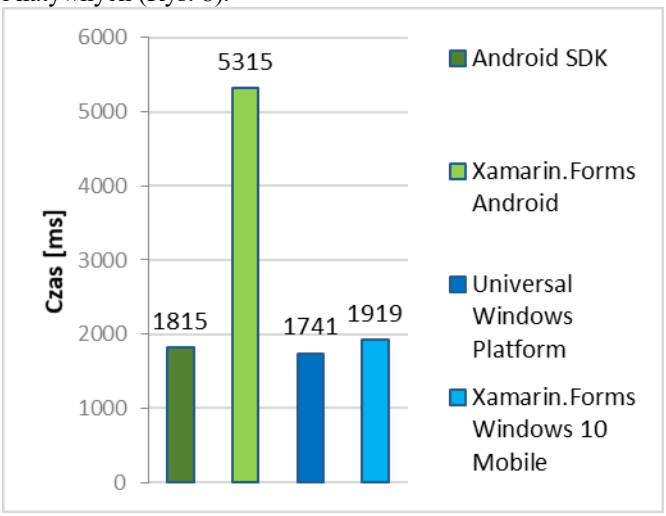


Rys. 7. Test - czas zapisu 1000 plików o rozmiarze 50kB

W systemie Windows 10 Mobile występuje podobna różnica między rozwiązaniem wieloplatformowym a natywnym jak w poprzednim teście zapisu dużego pliku. Wynosi ona 8% na niekorzyść aplikacji napisanej we frameworku Xamarin.Forms.

5.3.3. Czas odczytu pliku 50MB

W kolejnym eksperymencie został wykonany odczyt pliku o rozmiarze 50MB za pomocą aplikacji wieloplatformowych i natywnych (Rys. 8).



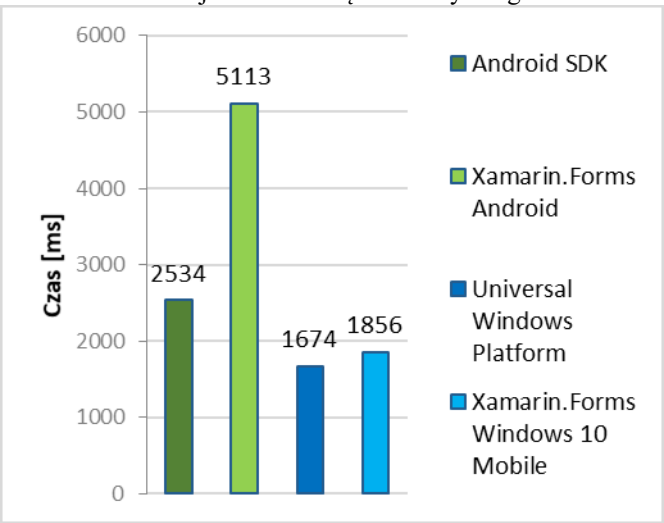
Rys. 8. Test - czas odczytu pliku 50MB

W przypadku systemu Android rozwiązanie Xamarin.Forms jest prawie trzykrotnie wolniejsze od rozwiązania przygotowanego w Android SDK. Na systemie mobilnym od Microsoftu różnica między rozwiązaniem wieloplatformowym a natywnym jest kolejny raz niewielka. Jak w poprzednich testach rozwiązanie to jest nieco wolniejsze od aplikacji natywnej, w tym wypadku o 10%.

5.3.4. Czas odczytu 1000 plików 50kB

Następny eksperyment polegał na zmierzeniu czasu odczytu 1000 plików o rozmiarze 50Kb (Rys. 9).

Podczas tego testu zmniejszyła się różnica pomiędzy Android SDK a Xamarin.Forms w porównaniu do odczytu dużego pliku. Rozwiązanie wieloplatformowe jest tylko dwukrotnie wolniejsze od rozwiązania natywnego.



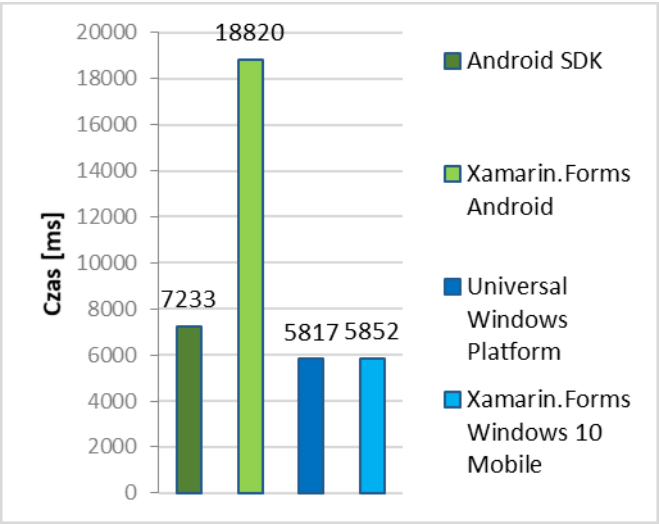
Rys. 9. Test - czas odczytu 1000 plików o rozmiarze 50kB

W przypadku systemu Windows 10 Mobile po raz kolejny występuje niewielka różnica na niekorzyść rozwiązania wieloplatformowego Xamarin.Forms, która podczas tego testu wynosi 11%, zatem spowolnienie działania jest porównywalne do pozostałych testów.

5.4. Test prędkości połączenia internetowego

Ostatni z testów polega na pobraniu pliku poprzez połączenie internetowe o rozmiarze 100MB (Rys. 10).

Po raz kolejny szybkość aplikacji wieloplatformowej na systemie mobilnym Android była dużo mniejsza od rozwiązania natywnego. Implementacja testu w Xamarin.Forms zakończyła zadania w czasie ponad dwa i pół razy dłuższym od analogicznej implementacji w Android SDK.



Rys. 10. Test - pobieranie pliku o rozmiarze 100MB

W przypadku systemu mobilnego od Microsoftu zadanie zostało zakończone zarówno dla rozwiązania natywnego jak i wieloplatformowego w podobnym czasie, różnica nie przekroczyła 1%.

6. Wnioski

Zebranie wyników badań na platformie Android wskazuje na wysokie spowolnienie działania spowodowane wykorzystaniem wieloplatformowego szkieletu aplikacji w 7 testach na 9. Na tym systemie operacyjnym jedynie w przypadku zapisu pliku rozwiązanie Xamarin.Forms było szybsze od rozwiązania natywnego. Największa różnica wydajności wystąpiła podczas używania funkcji skrótu SHA-256, gdzie implementacja natywna okazała się 28 razy szybsza.

Inaczej wygląda sytuacja na systemie mobilnym Windows 10 Mobile, gdzie występuje dużo mniejsza różnica między technologią natywną Universal Windows Platform a rozwiązaniem wieloplatformowym Xamarin.Forms. Większość testów wykazała różnicę w wydajności nieprzekraczającą 10% na niekorzyść wieloplatformowego szkieletu aplikacji, co wskazuje na zbliżoną wydajność obu rozwiązań. Używanie Xamarin.Forms zdecydowanie negatywnie wpływa jedynie na czas uruchamiania aplikacji, gdzie został on wydłużony o 66%.

Postawiona w artykule hipoteza badawcza mówiąca o tym, że tworzenie aplikacji ze wspólnym szkieletem umożliwia wydajne ich działanie niezależnie od systemu operacyjnego została obalona. Odpowiednia wydajność jest zapewniona jedynie w przypadku systemu Windows 10 Mobile, natomiast na urządzeniach pracujących pod kontrolą systemu operacyjnego Android korzystanie z wieloplatformowego szkieletu aplikacji w większości przypadków daje znaczne spowolnienie jej działania.

Literatura

[1] L. Delia, N. Galdamez, P. Thomas, L. Corbalan i P. Pesado, „Multi-platform mobile application development analysis," w Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on, IEEE, 2015.

[2] A. Troelsen, Język C# 2010 i platforma .NET 4, Warszawa: Wydawnictwo Naukowe PWN, 2011.

[3] J. Dickson, Xamarin Mobile Development, Allendale: Grand Valley State Univerity, 2013.

[4] P. Čečil, Cross-platform Mobile Development, Prague: Department of Software Engineering, 2015.

[5] D. Hermes, Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals, New York: Apress, 2015.

[6] C. Petzold, Creating Mobile Apps with Xamarin.Forms, Washington: Microsoft Press, 2016.

[7] https://www.xamarin.com/forms. [03. 06.2017].

[8] C. C. Sirvent Mazarico, Comparison between Native and Cross-Platform Apps, Växjö: Linnaeus University, Faculty of Technology, Department of Computer Science., 2015

[9] M. Reynolds, Xamarin Essentials, Birmingham: Packt Publishing, 2014.

[10] C. Bilgin, Mastering Cross-Platform Development with Xamarin, Birmingham: Packt Publishing, 2016.

[11] N. Panigrahy, Xamarin Mobile Application Development for Android - Second Edition, Birmingham: Packt Publishing, 2015.