JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 318-323
Wysłane: 2018-06-30
Przyjęte: 2018-07-03

Analiza porównawcza sposobów tworzeniu aplikacji dla systemu Android z wykorzystaniem technologii Xamarin

Michał Bartkiewicz*, Adrian Dziedzic

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

Streszczenie. Praca przedstawia analizę technologii Xamarin w dwóch trybach pracy: Xamarin.Forms oraz Xamarin.Native, które pozawalają na stworzenie aplikacji na urządzenia mobilne z systemem Android. Porównanie dotyczy liczby linii wygenerowanego kodu, wydajności poszczególnych elementów oraz rozmiaru zainstalowanej aplikacji i pliku instalacyjnego apk. Analiza została dokonana na podstawie dwóch takich samych aplikacji stworzonych oddzielnie w obu podejściach. w wyniku analizy wskazano bardziej efektywne rozwiązanie do wybranych zastosowań.

Słowa kluczowe: Xamarin; Android; C#; .NET

*Autor do korespondencji.

Adres e-mail: michal.bartkiewicz@pollub.edu.pl

Comparative analysis of approches in developing Android applications using Xamarin technology

Michał Bartkiewicz*, Adrian Dziedzic

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

Abstract. Article shows analysis of Xamarin technology in two modes: Xamarin Forms and Xamarin Native, used for developing applications for mobile devices with Android system. Comparison concerns the number of generated lines of code, performance of each part and size of installed application and size of apk installation file. Analysis was based on two identical applications created using both approaches. As a result of the analysis the more efficient approach for given purpose has been indicated.

Keywords: Xamarin; Android; C#; .NET

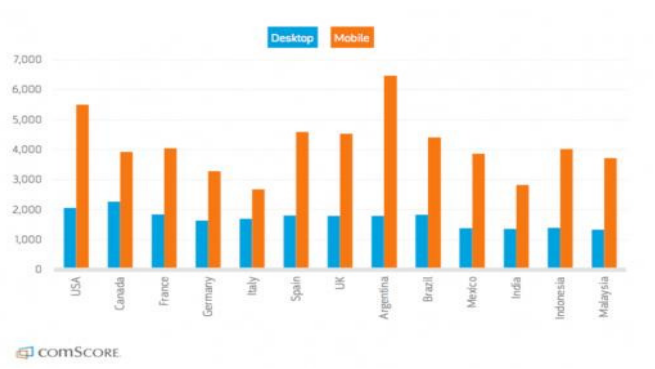
*Corresponding author.

E-mail address: michal.bartkiewicz@pollub.edu.pl

1. Wstęp

Dominującą częścią rynku z perspektywy wykorzystywanego oprogramowania przez zwykłych użytkowników stały się niewątpliwie smartfony. Urządzenia te nie służą już tylko do wysyłania wiadomości sms, czy dokonywania połączeń głosowych z innym odbiorcą, jak to miało miejsce w przypadku pierwszych telefonów komórkowych, ale umożliwiają wykonywanie wielu różnych czynności. Czynności te mogą być związane z pozyskiwaniem informacji, wykorzystaniem wbudowanych czujników i synchronizacją z innym sprzętem, czy też oprogramowaniem. Telefony mogą być również wykorzystywane do wspierania operacji biznesowych, takich jak dokonywanie płatności, a także służą celom czysto rozrywkowym. Niewątpliwą zaletą tych urządzeń, przyczyniającą się do popularności jest ich kompaktowy rozmiar, dzięki czemu każdy może mieć je zawsze pod ręką.

Rynek urządzeń mobilnych swoją popularnością przewyższył rynek urządzeń desktopowych. Użytkownicy więcej czasu spędzają z urządzeniami mobilnymi, do których zaliczają się również tablety, niż z komputerami osobistymi klasy PC [1]. Taka sytuacja spowodowała nagły wzrost zapotrzebowania na wytwarzanie oprogramowania, które jest dedykowane na urządzenia mobilne.



Rys. 1. Porównanie użycia urządzeń mobilnych oraz desktopowych przez użytkowników przy wykorzystaniu dłuższym niż 2 minuty [1]

Od niedawna rynek technologiczny dotyczący rozwiązań mobilnych się zmienił. Zaczęły pojawiać się nowe technologie, które często pozwalają na budowanie aplikacji na wiele różnych platform sprzętowych. Pozwala to na zaoszczędzenie czasu i pieniędzy, ponieważ stworzenie natywnych aplikacji na każdą platformę wymaga innego

podejścia i znajomości wielu technologii. Do rozwiązań cross-platformowych zalicza się również technologia Xamarin. Technologia Xamarin powstała w 2011 roku, a w 2016 została przejęta przez firmę Microsoft.

2. Xamarin

Technologia ta pozwala na tworzenie oprogramowania na wielu urządzeniach z różnymi systemami operacyjnymi. Wykorzystywana jest głównie do tworzenia aplikacji na telefony z systemem Android, Windows Mobile oraz IOS, ale jej możliwości są znacznie większe. Pozwala również na tworzenie aplikacji na telewizory, tablety oraz wszystkie urządzenia wspierane przez technologię UWP [2].

Technologia UWP (ang. Universal Windows Platform) umożliwia tworzenie aplikacji uniwersalnych na platformy o bardzo różnej wydajności sprzętowej oraz rozmiarze ekranu. Możliwe jest tworzenie aplikacji zaczynając od urządzeń wbudowanych aż do wielkich tablic interaktywnych Surface Hub oraz budowanie aplikacji na komputery z systemem Windows [3].

Główną zaletą technologii Xamarin jest możliwość pisania aplikacji z wykorzystaniem języka C# wykorzystując platformę programistyczną .NET. w obu podejściach jakie stosuje technologia Xamarin współdzieli się logikę biznesową, która wykorzystywana jest w każdym urządzeniu, na które buduje się program. Metody biznesowe tworzy się raz i są one dostępne dla każdej platformy [4].

W niniejszej pracy badania skupią się jedynie na urządzeniach mobilnych, wykorzystujących system Android, dlatego też dalsze opisy będą dotyczyły jedynie tego systemu. Są to Xamarin Forms i Xamarin Native.

2.1. Xamarin Forms

Xamarin Forms pozwala nie tylko współdzielić logikę biznesową, ale również graficzny interfejs użytkownika. Wiele platform składa się z bardzo podobnych komponentów, takich jak etykiety, pola tekstowe, przyciski, czy możliwość wyświetlania listy elementów. Xamarin Forms dostarcza wspólnego interfejsu, który zostaje odpowiednio tłumaczony na natywne rozwiązania [5].

Xamarin Forms wykorzystuje bardzo podobny styl tworzenia aplikacji, jak wcześniejsze technologie firmy Microsoft, czyli technologię Universal Windows Platform, czy starszą technologię Windows Presentation Foundation. W podanych rozwiązaniach do tworzenia interfejsu użytkownika wykorzystuje się pliki XAML. Są to pliki XML ze zdefiniowanymi atrybutami służącymi do tworzenia graficznych komponentów. Tak jak w plikach XML, tak pliki XAML mają strukturę drzewiastą, ponieważ dany element może się zawierać w innym elemencie [6].

Oprócz plików XAML, występują tutaj pliki z kodem w języku C#, w którym można tworzyć różne funkcje do programu. Każdy plik XAML jest powiązany z klasą C# odpowiadającą stronie graficznej, w której można w sposób programistyczny zarządzać elementami GUI. w aplikacji tworzonej w podejściu Forms został przyjęty właśnie taki styl pisania aplikacji, który wykorzystuje pliki XAML z powiązanymi plikami C#. Taki sposób pisania programów nosi nazwę wzorca widoku autonomicznego [7].

2.2. Xamarin Native

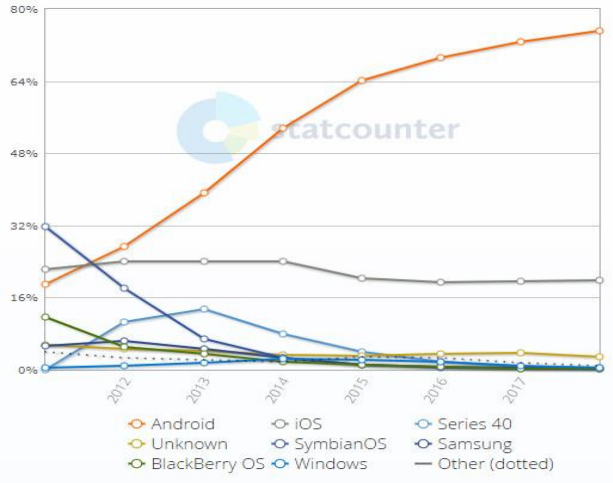
Drugie podejście pozwala na tworzenie natywnych aplikacji, dostosowanych pod każdą platformę z osobna. Nie ma tutaj, jak w przypadku Xamarin Forms, możliwości współdzielenia interfejsu użytkownika, ale cała logika biznesowa może być wykorzystywana na każdej platformie sprzętowej.

Architektura oraz sposób pisania aplikacji jest w zasadzie identyczny, jak podczas tworzenia tej aplikacji w ich macierzystych technologiach. Na systemie Android występują klasy aktywności wraz z ich cyklem życia, pliki XAML z graficznym interfejsem użytkownika, czy też usługi. Różnicą jest jedynie inne środowisko programistyczne oraz wykorzystywany język, wraz z wprowadzonymi konwencjami (np. nazwa metody rozpoczynająca się wielką literą) [8].

Podejście to pozwala pisać aplikacje bardziej złożone niż oferuje Xamarin Forms ze względu na wykorzystywanie API danej platformy, bez konieczności współdzielenia jej z innymi rozwiązaniami. Duża liczba rozwiązań dostępnych na system Android jest możliwa do implementacji podczas korzystania z Xamarin Native.

2.3. Android

Android w ostatnich latach sukcesywnie zdobywał popularność i obecnie jest to najbardziej popularny system operacyjny, wykorzystywany na urządzeniach typu smartfon. Jego udział w rynku to około 75 %, podczas gdy urządzenia z systemem iOS zajmujące drugie miejsce mają udział około 20%. Pozostałe systemy operacyjne straciły już na znaczeniu i stanowią niewielki procent rynku [9].



Rys. 2. Światowy udział systemów operacyjnych na urządzeniach typu smartphone [9]

Aplikacje w tym systemie od zawsze można tworzyć w języku Java, oraz od niedawna w języku Kotlin, stworzonym przez firmę JetBrains.

Android jest systemem dostarczanym na wielu różnych urządzeniach. W maju 2017 r. na konferencji Google I/O ogłoszono, że są ponad 2 miliardy aktywnych urządzeń z systemem Android.

3. Opis badania

Przy porównaniu technologii Xamarin Forms i Xamarin Native zostaną wzięte pod uwagę następujące czynniki:

- Wydajność aplikacji w przypadkach opisanych w rozdziale 3;
- Rozmiar zainstalowanej aplikacji;
- Rozmiar wygenerowanego pliku instalacyjnego apk;
- Liczba linii kodu w projekcie.

W artykule postawiono następujące hipotezy:

- H1: Aplikacje napisane w technologii Xamarin Native są wydajniejsze od tych napisanych w Xamarin Forms pod względem renderowania elementów widoku;
- H2: Aplikacje napisane w technologii Xamarin Forms wymagają porównywalnej liczby linii napisanego kodu, jeżeli rozważana jest tylko jedna platforma.

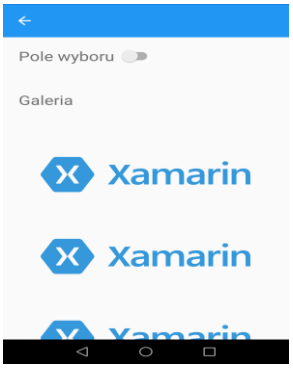
4. Opis aplikacji testowej

W celu przeprowadzenia badań zostały stworzone dwa projekty. Pierwszy z nich został przygotowany w rozwiązaniu Xamarin Forms, a drugi w Xamarin Native. Oba projekty zostały dostosowane jedynie dla systemu Android oraz zostały zbudowane na podstawie pustego szablonu podczas tworzenia projektu, dzięki czemu nie został wygenerowany nadmiarowy kod. Oba projekty posiadają taki sam wygląd i funkcjonalności, dzięki czemu można porównać parametry wydajnościowe oraz zobaczyć różnicę pomiędzy technologiami.

Aplikacja składa się z czterech stron graficznych. Pierwsza to strona powitalna służąca do nawigacji do odpowiednich testów dostępnych na innych ekranach (Rys. 3).

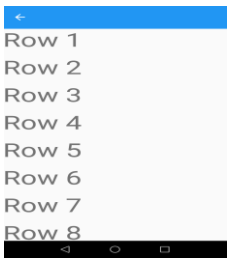
Rys. 3. Widok startowego ekranu testowanej aplikacji

Druga strona (wizytówka) składa się z prostych komponentów graficznych, wykorzystywanych w nowoczesnych aplikacjach mobilnych. Strona ta wyświetla takie elementy jak etykiety tekstów, pola tekstowe, przełączniki, przyciski oraz galerię. Cały ekran został dodatkowo ujęty w widoku ScrollView (element pozwalający na przesuwanie ekranu i podgląd obiektów, które się na nim nie mieszczą) [10]. Zostały tutaj wybrane jedynie komponenty dostępne na wszystkie platformy, żeby w prosty sposób dało się je zaimplementować w Xamarin Forms (Rys. 4).



Rys. 4. Widok ekranu wizytówki testowanej aplikacji, źródło wykorzystanego obrazka [11]

Trzecia strona to widok listy. Zastosowano tutaj kontrolki, które pomagają w pracy z tego typu danymi. w obu podejściach jest to kontrolka ListView (Rys. 5). Czwarta strona jest podobna do strony trzeciej z jedną różnicą - tutaj elementy listy zostały dodawane ręcznie, bez korzystania z wbudowanych komponentów kolekcji. Każdy wiersz listy został utworzony w sposób programistyczny, a następnie przypisany do widoku, tak aby wszystkie układały się wertykalnie (tak samo jak w stronie trzeciej). Dodatkowo całość strony jest skrolowana przez użycie wcześniej wspomnianego elementu ScrollView. Dla stron z listą danych test został przeprowadzony dla 1 000, 10 000 i 100 000 elementów.



Rys. 5. Widok ekranu listy rozwijanej testowanej aplikacji

5. Platforma testowa

Aplikacja została uruchomiona na urządzeniu o parametrach pokazanych w tabeli 1.

Tabela 1. Parametry urzadzenia testowego

Nazwa parametru	Wartość parametru
System operacyjny	Android
Wersja systemu operacyjnego	7.0
Nazwa urządzenia	Honor 8
Procesor CPU	Hisilicon Kirin 950
Pamięć RAM	4,0 GB
Pamięć urządzenia	32 GB
Rozdzielczość	1080 x 1920

6. Wyniki badań

Pierwszym etapem analizy było sprawdzenie wydajności po utworzeniu testowej aplikacji poprzez sprawdzenie czasu potrzebnego na załadowanie strony (Tabela 2). Początkowo testowana była strona wizytówka, a każdy test różnił się od siebie liczbą obrazów umieszczonych wewnątrz widoku.

Tabela 2. Czas ładowania strony wizytówki

Nazwa testu	Czas [s] w Xamarin Native	Czas [s] w Xamrin Forms
Strona wizytówka z 5 obrazkami	0,164	0,405
Strona wizytówka z 50 obrazkami	0,166	0,430
Strona wizytówka z 500 obrazkami	0,204	1,399
Strona wizytówka z 2000 obrazkami	0,317	10,048

Kolejne badanie przeprowadzone było na liście generowanej automatycznie. Lista ta zawierała różną liczbę wierszy w każdym teście. w obu przypadkach wykorzystywany był komponent ListView (Tabela 3).

Tabela 3. Czas ładowania strony z kontrolką ListView

Liczba wierszy	Czas [s] w Xamarin Native	Czas [s] w Xamrin Forms
1 000 wierszy	0,144	0,221
10 000 wierszy	0,400	0,252
100 000 wierszy	2,843	0,464
1 000 000 wierszy	27,832	2,317
10 000 000 wierszy	284,793	21,629

Ostatnie badanie przeprowadzane było również na generowaniu listy elementów, jednak teraz były one dodawane ręczenie (programowo) do layoutu, układającego elementy jeden pod drugim. Wyniki prezentuje tabela 4.

Tabela 4. Czas ładowania strony z generowaną ręcznie listą

Liczba wierszy	Czas [s] w Xamarin Native	Czas [s] w Xamrin Forms
1 000 wierszy	0,389	1,9
10 000 wierszy	2,912	112
50 000 wierszy	-	-
100 000 wierszy	-	-

Powyższa analiza przeprowadzona była również dla 50 000 oraz 100 000 wierszy, jednak czas oczekiwania na listę w podejściu Forms był większy niż pół godziny i badania zostały przerwane.

Analiza rozmiaru aplikacji zarówno po zainstalowaniu, jak przygotowaniu do procesu instalacji w postaci pliku apk.

Ostatni test związany był z liczbą linii kodu potrzebnego do napisania testowych aplikacji. Platforma GitHub, na której przechowywane były projekty w postaci dwóch oddzielnych prywatnych repozytoriów pozwala na odczytanie tych informacji, wliczając w to różne wygenerowane dodatkowo elementy (tabela 6).

Tabela 5. Rozmiar aplikacji

Nazwa testu	Rozmiar [MB] w Xamarin Native	Rozmiar [MB] w Xamrin Forms
Rozmiar aplikacji po utworzeniu projektu w trybie debbuger	19,50	23,42
Rozmiar aplikacji po ukończeniu w trybie debbuger	19,43	23,51
Rozmiar aplikacji po utworzeniu projektu w trybie release	15,30	18,01
Rozmiar aplikacji po ukończeniu w trybie release	15,32	18,11
Rozmiar apk po ukonczeniu aplikacji	14,2	17,5
Rozmiar apk po utworzeniu projektu	14,2	17,5

Tabela 6. Liczba linii kodu potrzebnej do utworzenia aplikacji

Nazwa testu	Liczba linii w Xamarin Native	Liczba linii w Xamrin Forms
Liczba linii kodu po wygenerowaniu projektu	731	7727
Liczba linii kodu po ukończeniu projektu	7247	10029
Liczba linii kodu potrzebna na implementacje	301	198

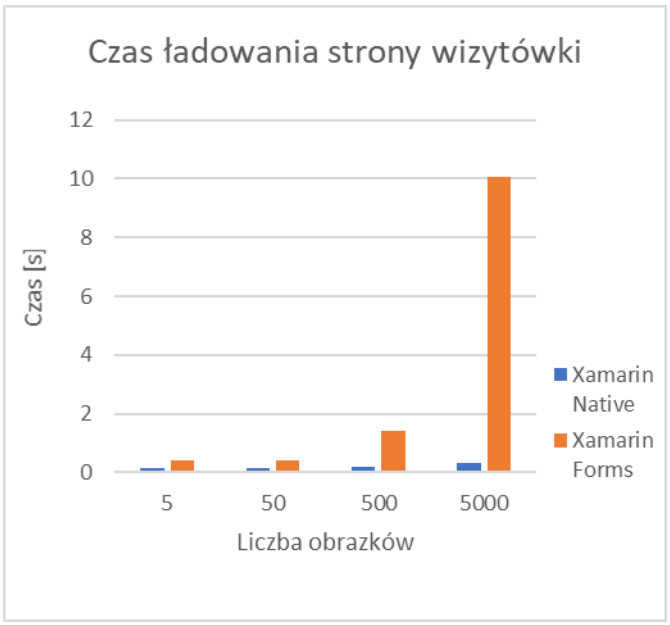
7. Analiza wyników

Pierwszy test polegał na zmierzeniu czasu wczytywania strony zawierającej różną liczbę obrazków. Wyniki jednoznacznie wskazują, że Xamarin Native jest szybszy – zarówno przy małej i dużej liczbie obrazków. Im więcej obrazów, tym większa przewaga. Xamarin Native poradził sobie bez problemu nawet z 2000 obrazków w akceptowalnym czasie – 317 milisekund podczas gdy Xamarin Forms potrzebował ponad 10 sekund (Rys. 6).

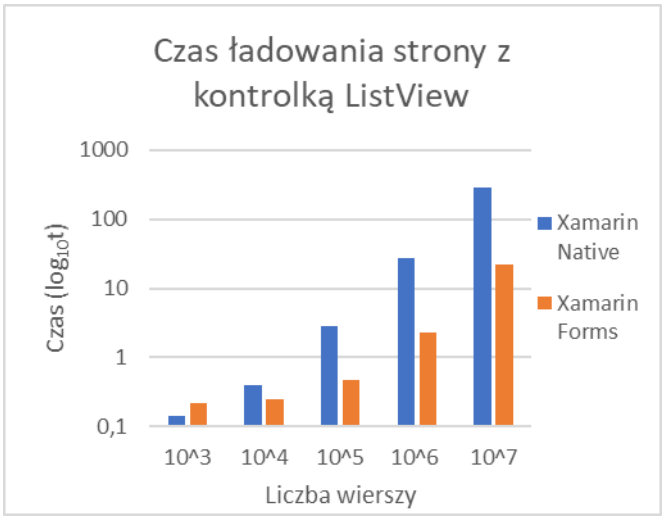
Drugi test polegał na porównaniu szybkości wczytywania listy z danymi. Przy 1000 elementów Xamarin Native okazał się szybszy, natomiast przy większej liczbie elementów znacznie lepiej wypada Xamarin Forms. Należy jednak zaznaczyć, że testowana była kontrolka ListView, która dla natywnej aplikacji androida nie jest już zalecana na rzecz wykorzystywania wydajniejszego komponentu RecyclerView [12] (Rys. 7).

Badania przeprowadzone na potrzeby artykułu „Performance analysis of native and cross-platform mobile applications" [4] również dotyczyły wydajności Xamarin Native i Xamarin Forms, ale skupiono się na innych aspektach aplikacji. Porównano wydajność obliczeniową, czas odczytu plików, szybkość pobierania obrazków oraz czas wyznaczania położenia. Eksperyment nie wskazał jednoznacznie wydajniejszego rozwiązania - Xamarin Native okazało się szybsze w przypadku pobierania obrazków i odczycie oraz zapisie plików, natomiast Xamarin Forms było

szybsze w wyznaczaniu położenia i obliczeniach. Różnice pomiędzy podejściami okazały się mniejsze niż w tym eksperymencie, gdzie wykorzystanie Xamarin Native okazało się znacznie wydajniejsze w większości przeprowadzonych testów [4].

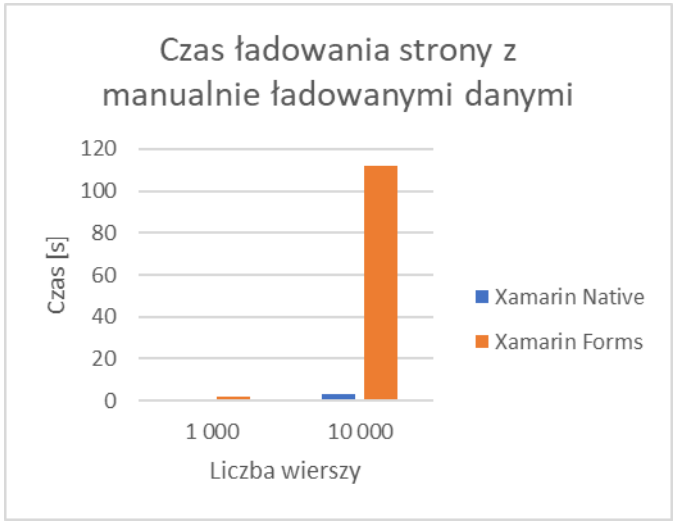


Rys. 6. Wykres pomiaru czasu załadowania strony z wizytówką



Rys. 7. Wykres w skali logarytmicznej pomiaru czasu załadowania strony z kontrolką ListView

Kolejny test polegał na zmierzeniu czasu wczytywania listy przy ręcznym dodawaniu jej elementów. Sposób ten okazał się dużo wolniejszy przy obu podejściach, natomiast znacznie lepiej wypadł Xamarin Native. Przy 1000 wierszy na wczytanie wystarczyło niecałe 400 milisekund, co jest czasem akceptowalnym, natomiast Xamarin Form potrzebował prawie 2 sekundy (Rys. 8). Ze względu na szybki wzrost czasu ładowania wraz ze wzrostem liczby elementów testy dla 50 000 i 100 000 wierszy nie zostały przeprowadzone. Już przy 10 000 wierszy czasy przestały być akceptowalne - w przypadku Xamarin Native były to prawie 3 sekundy, a w przypadku Xamarin Forms prawie 2 minuty. Dziesięciokrotnie większej liczby wierszy nie udało się załadować w obu podejściach, dla Xamarin Native system Android wyłączył aplikację po paru minutach od próby wczytania. Dla Xamarin Forms aplikacja próbowała wczytać dane przez ponad godzinę, jednak bez żadnego skutku - widok nie załadował się. Należy również wspomnieć, że w podejściu Xamarin Forms, dla 100 000 wierszy, podczas przewijania występowały odczuwalne przycięcia, w drugim podejściu nie było takiego problemu.



Rys. 8. Wykres pomiaru czasu załadowania strony z manualnie dodawanymi wierszami

Ostatnie porównanie dotyczyło wykorzystania miejsca w pamięci przez aplikacje. w każdym przypadku rozmiar aplikacji w Xamarin Forms był większy o około 3 MB, co stanowi około 20% więcej zajętej pamięci niż w przypadku aplikacji Xamarin Native.

Mniejsza liczba linii kodu przemawia za podejściem Xamarin Forms, jednak należy zaznaczyć, że tworzenie widoków w XAML jest dużo bardziej przejrzyste i wymaga mniejszej liczby linii kodu. Kontrolki zdefiniowane w podejściu Xamarin Native często wymagają więcej zajętego miejsca, ponieważ atrybuty są dłuższe i należy pisać je jeden pod drugim, aby kod był czytelny. w XAML atrybuty kontrolki nie mają takich długich nazw i można je wypisać po kilka w jednej linijce. Jednak aplikacja ta jest stosunkowo mała, a Xamarin Forms zajął około 100 linijek kodu mniej, co oznacza prawię 1/3 całości.

8. Wnioski

Większość wyników zdecydowania przemawia na korzyść Xamarin Native, zarówno pod względem wydajności, jak i zajmowanej pamięci. Pozwala on na szybsze wyrenderowanie widoków oraz płynniejszą pracę z dużą liczbą komponentów. w Xamarin Forms czas potrzebny na załadowanie danych czasami jest odczuwalny przez użytkownika systemu, a niektóre elementy się przycinają.

Z kolei Xamarin Forms radzi sobie lepiej, jeżeli mowa o szybszym tworzeniu aplikacji. Pozwala on pisać aplikacja z mniejszą ilością kodu, a dodatkowo uruchamiać aplikację na wielu platformach sprzętowych. w tworzeniu aplikacji tylko na jeden dedykowany system nie ma dużej różnicy

pomiędzy wykorzystywanym kodem, ponieważ oba rozwiązania korzystają z tych samych modułów przy wykonywaniu operacji na modelu danych. Różnicą jest natomiast sposób definiowania widoków, jednak i to zależy od sposobu implementacji przez programistę. Natomiast dla wielu platform sprzętowych, różniących się od siebie, Xamarin Forms pozwolił by zredukować znacznie ilość kodu, co zapewniło by szybszą implementację aplikacji.

Literatura

[1] https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ [22.06.18]

[2] C. Petzold, Creating Mobile Apps with Xamarin.Forms, Microsoft Press, 2016

[3] https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide [23.06.18]

[4] P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik, J. Smołka, Performance analysis of native and cross-platform mobile applications, Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 2017 T. 7, nr 2, str. 50--53

[5] G. Taskos, Xamarin Cross-Platform Development Cookbook, Packt Publishing, 2016

[6] J. Peppers, G. Taskos, C. Bilgin, Xamarin: Cross-Platform Mobile Application Development, 2016

[7] J. Matulewski, MVVM i XAML w Visual Studio 2015, helion 2016

[8] Mark Reynolds, Xamarin Mobile Application Development for Android, Packt Publishing, 2014

[9] http://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2011-2018 [22.06.18]

[10] J. Morris, Android User Interface Development Beginner's Guide, Packt Publishing, 2011

[11] https://ermlab.com/2015/10/22/xamarin-forms-przykladowa-aplikacja/ [22.06.18]

[12] https://docs.microsoft.com/en-us/xamarin/android/user-interface/layouts/list-view/ [22.06.18]