JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 183-190
Wysłane: 2017-12-14
Przyjęte: 2017-12-19

Analiza możliwości zastosowania platformy Xamarin do budowy aplikacji wieloplatformowych mobilnych

Michał Dras*, Grzegorz Fila, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono platformę Xamarin wykorzystywaną do tworzenia aplikacji wieloplatformowych na systemy: Android, iOS, MacOS oraz platformę Universal Windows Platform. Omówiona została sama platforma, a także aplikacja testowa oraz badania przeprowadzone w celu przeanalizowania możliwości platformy i jej efektywności w budowaniu aplikacji. Badania dotyczyły części wspólnej kodu, rozmiaru aplikacji po zainstalowaniu oraz jej wydajności. Pokazały one, że z pomocą Xamarina możliwe jest tworzenie aplikacji o części wspólnej kodu powyżej 80%, przy zachowaniu niewielkiego rozmiaru i zadowalającej wydajności.

Słowa kluczowe: xamarin; aplikacje wieloplatformowe; systemy mobilne; c#

*Autor do korespondencji.

Adres e-mail: dras.michael@gmail.com

Analysis of Xamarin capabilities for building mobile multi-platform applications

Michał Dras*, Grzegorz Fila, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents Xamarin platform which is used to create cross-platform application for Android, iOS, MacOS and Universal Windows Platform. This article shows Xamarin platform and a test application that has been used to investigate platform's capabilities and effectiveness in creating multi-platform applications. Inspections prove that Xamarin allows to create multi-platform applications in a more effective way without losing too much on performance of these applications on individual platforms and systems.

Keywords: xamarin; multi-platform applications; mobile systems; c#

*Corresponding author.

E-mail address: dras.michael@gmail.com

1. Wstęp

Obecnie aplikacje mobilne są coraz popularniejsze. Jest to nieodzownie związane z ciągle zwiększającym się udziałem smartfonów w rynku urządzeń mobilnych. Wśród smartfonów największy udział mają urządzenia z systemem Android. Drugie miejsce zajmuje system iOS, natomiast trzecie Windows 10 Mobile. Każdy z wyżej wymienionych systemów wymaga oddzielnie pisanej aplikacji: Android wymaga aplikacji napisanej w Javie, iOS w Objective-C, a Windows 10 Mobile w C#. Tworzenie takiej samej aplikacji trzy razy jest mało efektywne. W celu rozwiązania tego problemu powstały aplikacje wieloplatformowe, które są pisane raz, ale działają na wielu platformach. Przykładem platformy wykorzystywanej do budowania aplikacji wieloplatformowych jest Xamarin [1].

Zdaniem autorów [2] wydajność aplikacji stworzonych z pomocą Xamarina jest porównywalna do wydajności aplikacji natywnych. Posiada on także dużą liczbę bibliotek, skracających czas pisania aplikacji. Autor [3] twierdzi z kolei, że Xamarin pozwala wykorzystać w pełni możliwości API platform docelowych oraz wszystkie zalety języka C#.

W celu przeanalizowania możliwości platformy Xamarin przy tworzeniu wieloplatformowych aplikacji mobilnych powstała aplikacja testowa i przeprowadzona została seria badań. Mają one wykazać prawdziwość hipotezy, która mówi, że dzięki zastosowaniu Xamarina, możliwe jest uzyskanie dużej części kodu wspólnego dla wszystkich platform docelowych i zachowanie przy tym zadowalającej wydajności aplikacji.

2. Xamarin

Xamarin jest platformą wykorzystywaną do tworzenia aplikacji wieloplatformowych przy pomocy języka C#. Platforma ta została stworzona przez firmę o tej samej nazwie, która została założona przez Miguela de Icaza 16 maja 2011 roku. Nazwa Xamarin wywodzi się od nazwy gatunku małpki - tamaryny, natomiast pierwsza litera jest nawiązaniem do poprzedniej platformy stworzonej przez firmę tego samego założyciela - Ximian. Xamarin został przejęty przez Microsoft 24 lutego 2016 roku. Od tego czasu Xamarin jest dalej rozwijany, a Xamarin SDK został udostępniony na zasadzie open-source.

Xamarin powstał w oparciu o architekturę Windows Runtime, co pozwala na budowanie aplikacji Universal Windows Platform bez użycia dodatkowych bibliotek natywnych [4]. Aplikacje pisane na pozostałe dwa systemy, czyli Android i iOS, wymagają bibliotek natywnych, które są

dostępne z poziomu Xamarina. Są to Xamarin.Android oraz Xamarin.iOS.

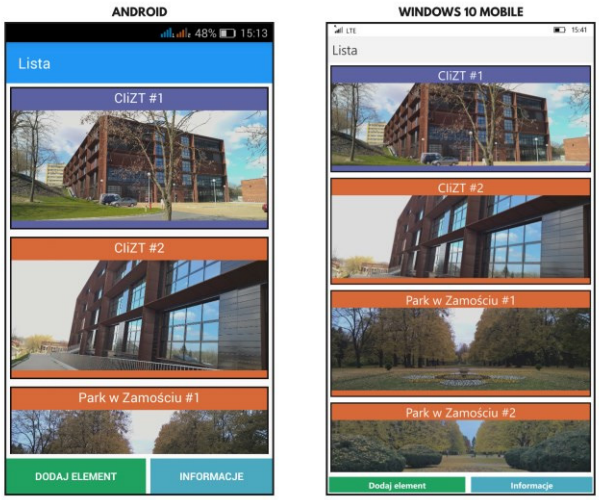
Jak już wcześniej wspomniano, aplikacje muszą być pisane przy pomocy jednego języka, którym jest C#. Aplikacje takie nie będą działały na Androidzie i iOS ponieważ platformy te wymagają innego języka. W celu zapewnienia działania aplikacji napisanych przy pomocy Xamarina, stworzone zostały dwa sposoby ich przystosowania do konkretnej platformy [5].

Pierwszym ze sposobów jest mechanizm zaprojektowany dla potrzeb Androida. Aplikacja w trakcie działania wprowadza komunikację między Javą a platformą .Net. Komunikacja ta zapewnia działanie aplikacji dla platformy .Net na platformie Java.

Drugim sposobem, przeznaczonym dla potrzeb iOS-a, jest prekompilowanie kodu do kodu wykorzystywanego przez system docelowy – Objecive-C. W przypadku tego systemu, aplikacja nie może nawiązywać komunikacji z systemem, jeżeli nie jest ona stworzona w natywnym języku.

3. Aplikacja testowa

W celu przeprowadzenia badań, w środowisku Xamarin stworzona została aplikacja na systemy Android i Windows 10 Mobile (UWP). Jej główną funkcją jest budowanie listy zdjęć (galerii) o możliwościach CRUD (Create, Read, Update, Delete) (Rys. 1). Możliwe jest zatem dodawanie nowych elementów, a także przeglądanie, edytowanie i usuwanie już istniejących.



Rys. 1. Strona główna aplikacji testowej

3.1. Interfejs

W tworzeniu aplikacji wykorzystano API Xamarin.Forms. Umożliwia on wykonanie jednego wspólnego interfejsu dla wszystkich platform docelowych. Stosuje się do tego języki XAML lub C# [6].

Xamarin.Forms posiada wbudowane kilka typów stron i układów, a także ponad 20 rodzajów kontrolek, które można wykorzystać do budowy interfejsu. Każdy z tych elementów musi mieć swojego odpowiednika na każdej z platform docelowych. W czasie działania aplikacji elementy pochodzące z Xamarin.Forms są mapowane właśnie na elementy natywne, odpowiednie dla danego systemu [7].

Zdarza się, że kontrolki natywne posiadają parametry, których nie da się zmodyfikować we wspólnej części kodu. Rozwiązaniem może być wówczas stworzenie niestandardowej kontrolki (Przykład 1), dziedziczącej po jednej z kontrolek Xamarin.Forms oraz rendererów dla każdej z platform docelowych (Przykład 2). To właśnie renderer określa sposób wyświetlania elementu na danej platformie [8].

Przykład 1. Przykładowy kod kontrolki niestandardowej

```
using Xamarin.Forms;
namespace Apka
{
public class CustomButton : Button
{
}
}
```

Przykład 2. Przykładowy kod renderera

```
[assembly: ExportRenderer(typeof(CustomButton),
typeof(CustomButtonRenderer))]
namespace Apka.Droid
{
public class CustomButtonRenderer : ButtonRenderer
{
protected override void
OnElementChanged(ElementChangedEventArgs<Button> e)
{
base.OnElementChanged(e);
if(Control != null)
{
Control.SetPadding(5, 2, 5, 2);
}
}
}
}
```

Xamarin.Forms pozwolił uczynić wspólną większość kodu odpowiedzialnego za interfejs aplikacji. Każda z platform wymaga jedynie wywołania metody inicjalizującej API Xamarin.Forms oraz utworzenia klas odpowiednich rendererów, w przypadku stosowania niestandardowych kontrolek.

3.2. Przechowywanie danych

W celu zachowania danych po zamknięciu aplikacji, zaimplementowano w niej funkcję zapisu listy elementów do pliku (Przykład 3). Najpierw dane są serializowane do formatu JSON, następnie wybierany jest folder i tworzony jest plik, w którym informacje te zostaną umieszczone [9].

Przykład 3. Metoda zapisująca listę elementów do pliku

```
public static async void saveJSON()
{
var d = JsonConvert.SerializeObject(Elements.AllElements);
IFolder rootFolder = FileSystem.Current.LocalStorage;
IFolder folder = await
rootFolder.CreateFolderAsync("ApkaData",
CreationCollisionOption.OpenIfExists);
IFile file = await folder.CreateFileAsync("elements.json",
CreationCollisionOption.ReplaceExisting);
await file.WriteAllTextAsync(d);
}
```

Skorzystano tu z frameworka Newtonsoft.Json oraz API PCLStorage [10]. Umożliwiły one zrealizowanie operacji zapisu i odczytu danych z całkowicie wspólnym kodem.

3.3. Przechwytywanie obrazów

Podczas dodawania elementu, zdjęcia przechwytywane są z aparatu urządzenia lub pobierane z pamięci wewnętrznej. Można do tego wykorzystać dowolną aplikację do obsługi aparatu lub przeglądania plików [11].

Do zrealizowania tej funkcji posłużyła wtyczka MediaPlugin. Kod jest w pełni współdzielony. Wymagane jest jedynie nadanie odpowiednich uprawnień.

3.4. Lokalizacja

Podczas dodawania nowego elementu do listy można także pobrać aktualną lokalizację z wykorzystaniem GPS-u lub połączenia internetowego i automatycznie uzupełnić jedno z pól formularza.

Najpierw, przy pomocy wtyczki GeolocatorPlugin, pobierane są współrzędnie geograficzne, a następnie na ich podstawie określany jest prawdopodobny adres miejsca, w którym znajduje się użytkownik (tzw. reverse geocoding) [12].

Wykorzystywane jest do tego API Xamarin.Forms.Maps, które w celu ustalenia adresu korzysta z usług właściwych platformie docelowej (Android – Google Maps, Windows 10 Mobile – Bing Maps) [13].

Na wypadek gdyby Xamarin.Forms.Maps nie udało się określić adresu, stworzono alternatywne rozwiązanie. Jest nim klasa CustomGeocoder.cs (Przykład 4), która reverse geocoding wykonuje z pomocą API Google Maps i zapytań HTTP [14].

Przykład 4. Klasa wykonująca reverse geocoding z wykorzystaniem API Google Maps i zapytań HTTP

```
public class CustomGeocoder
{
private string key = "<klucz_do_API_Google_Maps>";
public async Task<IEnumerable<string>>
ReverseGeocoding(double latitude, double longitude)
{
HttpClient client = new HttpClient();
var request =
string.Format(@"https://maps.googleapis.com/maps/api/geo
code/json?latlng={0},{1}&key={2}&language=pl", latitude,
longitude, key);
var response = await client.GetStringAsync(request);
var results = JObject.Parse(response)["results"];
List<string> list = new List<string>();
foreach (JToken i in results)
{
list.Add((string)i["formatted_address"]);
}
return list.AsEnumerable();
}
}
```

Poza wywołaniem w kodzie każdej z platform metody inicjalizującej Xamarin.Forms.Maps oraz, w przypadku Androida, dodaniem w manifeście informacji o kluczu do API Google Maps, reszta kodu jest wspólna. Niezbędne jest jednak przypisanie odpowiednich uprawnień, by aplikacja mogła korzystać z usług lokalizacyjnych.

3.5. Dane o urządzeniu

Jedną z pobocznych funkcji aplikacji jest wyświetlanie informacji o urządzeniu oraz odczytów z sensorów. Podane są tu: typ urządzenia, model, system operacyjny oraz jego wersja, stan baterii, jej status, a także dostępność połączenia internetowego. Sensory, z których można uzyskać odczyty, to z kolei: akcelerometr, żyroskop, magnetometr, kompas, czujnik orientacji, czujnik oświetlenia.

W realizacji tej funkcji pomogły następujące wtyczki: Plugin.DeviceInfo, Plugin.Battery, Plugin.Connectivity oraz Plugin.Sensors. Całość zrealizowana została w pełni wieloplatformowo.

3.6. Filtry graficzne

Do jednej ze stron aplikacji, wyświetlającej szczegółowe informacje o danym elemencie, dodano możliwość wyświetlania zdjęcia z zastosowaniem jednego z dostępnych filtrów. Są to Grayscale, Sepia, Invert Color oraz Black & White (Rys. 2). Nakładane są one na obrazy załadowane do pamięci podręcznej.

Rys. 2. Tryby wyświetlania obrazu

Do tego celu wykorzystano bibliotekę FFImageLoading, która służy do wczytywania i wyświetlania obrazów z różnych źródeł i ma do tego przeznaczone własne kontrolki, mogące zastąpić te, które pochodzą z Xamarin.Forms [15]. Skorzystanie z niej wymaga dodania wywołania metody inicjalizującej do części kodu każdej z platform docelowych. Pozostała część kodu jest wspólna dla każdego systemu.

4. Metody i przebieg badań

Zbadane zostały: część wspólna kodu źródłowego, rozmiar aplikacji po zainstalowaniu oraz jej wydajność.

W celu zbadania części wspólnej kodu aplikacji wykorzystano metrykę linii kodu źródłowego, będącą najprostszą metryką rozmiaru oprogramowania [16]. Pomogła w tym aplikacja LocMetrics, zliczająca liczbę linii kodu w wybranych katalogach. Wśród informacji, które zwraca, są: liczba linii fizycznych kodu źródłowego oraz liczba linii logicznych kodu źródłowego. Ta pierwsza oznacza całkowitą liczbę linii kodu z wyłączeniem linii pustych oraz linii składających się jedynie z komentarza. Druga natomiast dotyczy wyłącznie linii wpływających na logikę aplikacji, zawierających instrukcje. Pomijane są w niej zatem linie fizyczne kodu źródłowego, które zawierają jedynie nawias klamrowy a instrukcje zawarte w kilku liniach są liczone jako jedna. Dla języków, których aplikacja nie wspiera, takich jak XAML, linie logiczne kodu nie są zliczane. W związku z tym, że definiujące interfejs graficzny, pliki XAML stanowią znaczną część kodu współdzielonego, w tym konkretnym przypadku należy mieć ograniczone zaufanie do tej wartości.

Badanie rozmiaru aplikacji polegało na jej zainstalowaniu i odczytaniu rozmiaru z jednej ze stron w ustawieniach urządzenia.

Dla sprawdzenia wydajności przeprowadzone zostały pomiary czasu trwania najważniejszych funkcji aplikacji:

1) zapisu – składającego się z serializacji listy obiektów do formatu JSON i zapisania tych danych w pliku tekstowym,
2) odczytu – czyli wczytania danych z pliku tekstowego, deserializacji ich oraz dodania uzyskanych elementów do listy,
3) wczytywania i wyświetlania obrazu,
4) dodawania, modyfikowania i usuwania elementów listy,
5) pobierania współrzędnych geograficznych za pomocą GPS-u.

Czas trwania zapisu i odczytu został rozpatrzony dla listy składającej się z 1, 10, 100 oraz 1000 elementów. Wczytywanie i wyświetlanie obrazu zostało przetestowane dla plików o rozmiarach 0,5 MB, 1 MB, 2 MB, 5 MB z wyróżnieniem dwóch przypadków – bez użycia filtru oraz z zastosowaniem filtru Invert Color. Dla każdego przypadku przeprowadzono 10 pomiarów. Na ich podstawie wyznaczona została średnia wartość oraz odchylenie standardowe.

Fragment kodu wykorzystany do ustalenia czasu trwania poszczególnych operacji przedstawia przykład 5.

Przykład 5. Fragment kodu służący do pomiaru czasu wykonania badanych funkcji

```
DateTime startTime = DateTime.Now;
<fragment_kodu_którego_czas_wykonania_badamy>
DateTime stopTime = DateTime.Now;
TimeSpan roznica = stopTime - startTime;
await DisplayAlert("Wykonano", "Operacja trwała " +
roznica.TotalMilliseconds + " milisekund.", "OK");
```

Urządzenia, na których zostały przeprowadzone pomiary to:

- myPhone Luna (Android),
- Nokia Lumia 930 (Windows 10 Mobile).

5. Rezultaty badania

5.1. Część wspólna kodu

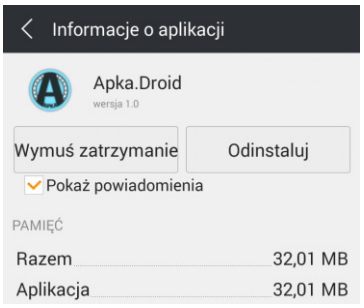
Pierwsze badanie dotyczyło części wspólnej kodu źródłowego aplikacji. Wyniki zostały zebrane w tabeli 1.

Tabela 1. Wyniki pomiarów liczby linii kodu

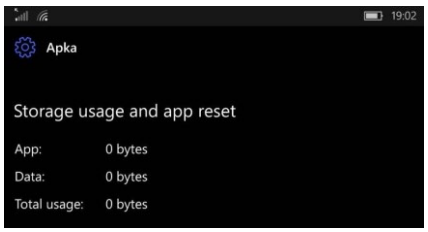
		Wspólne	Android	UWP	Suma
Pliki źródłowe		24	4	6	34
Linie kodu		1365	143	156	1664
Linie fizyczne kodu	liczba	1248	118	101	1467
	udział	85%	8%	7%	100%
Linie logiczne kodu	liczba	546	67	53	666
	udział	82%	10%	8%	100%

5.2. Rozmiar aplikacji

Zbadano również rozmiary aplikacji. Po zainstalowaniu na urządzeniu z systemem Android, aplikacja zajmuje 32 MB (Rys. 3). Niestety dla systemu Windows 10 Mobile nie udało się uzyskać takich informacji (Rys. 4).



Rys. 3. Informacje o aplikacji zainstalowanej na urządzeniu z systemem Android



Rys. 4. Informacje o aplikacji zainstalowanej na urządzeniu z systemem Windows 10 Mobile

5.3. Wydajność aplikacji

Badanie wydajności aplikacji składa się z pięciu części. Pierwsza z nich dotyczy czasu trwania operacji zapisu. Jej wyniki znajdują się w tabelach 2 i 3.

W drugiej mierzony jest czas trwania operacji zapisu. Wyniki zestawione są w tabelach 4 i 5.

Trzecia część, której wyniki zawarto w tabelach od 6 do 9, bada czas wczytywania obrazu, z wyróżnieniem dwóch przypadków – z zastosowaniem filtru oraz bez.

Czwarta bada czas wykonywania podstawowych operacji na elementach listy, czyli dodawania, modyfikowania oraz usuwania. Wyniki tych pomiarów przedstawiono w tabelach 10 i 11.

Piąta i ostatnia część badania wydajności mierzy czas potrzebny do uzyskania współrzędnych geograficznych z wykorzystaniem GPS-u. Jej rezultaty przedstawia tabela 12.

Tabela 2. Wyniki pomiarów czasu trwania operacji zapisu na platformie Android

		1 el. [ms]	**10 el. [ms]**	**100 el. [ms]**	**1000 el. [ms]**
Pomiar	1.	0,432	0, 723	2,974	21,851
	2.	0,465	0,666	3,167	22,420
	3.	0,425	1,583	2,877	22,531
	4.	0,412	0,651	2,858	22,378
	5.	0,428	0,661	2,916	23,146
	6.	0,433	0,872	2,825	22,265
	7.	0,426	2,628	2,704	22,424
	8.	0,415	0,649	5,501	26,303
	9.	0,431	0,658	2,732	23,047
	10.	0,440	0,661	3,373	22,868
Średnia		0,4307	1,003222222	3,1927	22,9233
Odchylenie standardowe		0,0146367	0,680985275	0,835582	1,248438

Tabela 3. Wyniki pomiarów czasu trwania operacji zapisu na platformie Windows 10 Mobile

		1 el. [ms]	**10 el. [ms]**	**100 el. [ms]**	**1000 el. [ms]**
Pomiar	1.	3,9965	15,6255	35,4962	312,5101
	2.	2,9987	15,6334	34,9970	315,4994
	3.	3,4945	15,6218	37,4948	316,5297
	4.	2,9961	15,6227	36,0075	312,5032
	5.	3,9967	15,6240	36,9962	311,4941
	6.	3,0008	15,6264	36,0194	321,0005
	7.	2,9962	9,9963	35,9946	312,0176
	8.	3,0019	15,4961	35,9967	312,0266
	9.	2,9967	15,6328	31,9961	311,0093
	10.	3,0044	5,4990	35,4960	312,5258
Średnia		3,24825	14,0378	35,64945	313,7116
Odchylenie standardowe		0,423575	3,4807864	1,473602	3,10026

Tabela 4. Wyniki pomiarów czasu trwania operacji odczytu na platformie Android

		1 el. [ms]	**10 el. [ms]**	**100 el. [ms]**	**1000 el. [ms]**
Pomiar	1.	1654,906	1688,741	1669,456	1751,026
	2.	1647,045	1632,476	1654,561	1764,562
	3.	1668,891	1659,274	1672,452	1773,378
	4.	1654,799	1659,415	1699,332	1726,952
	5.	1673,361	1648,650	1626,493	1725,662
	6.	1645,767	1665,450	1657,700	1747,956
	7.	1661,355	1639,579	1650,282	1759,079
	8.	1645,569	1637,717	1649,915	1744,079
	9.	1649,144	1681,694	1655,540	1759,636
	10.	1669,651	1660,462	1695,999	1743,724
Średnia		1657,0488	1657,346	1663,173	1749,605
Odchylenie standardowe		10,6195012	18,42991	22,01099	15,40394

Tabela 5. Wyniki pomiarów czasu trwania operacji odczytu na platformie Windows 10 Mobile

		1 el. [ms]	10 el. [ms]	100 el. [ms]	1000 el. [ms]
Pomiar	1.	68,9995	77,8771	105,5278	492,1302
	2.	65,0024	62,7341	107,7692	517,0285
	3.	64,5440	62,7782	107,5160	490,4102
	4.	64,0228	78,1275	110,0255	499,5718
	5.	64,0035	62,1838	106,5323	498,7774
	6.	66,5266	62,4927	107,7483	499,3605
	7.	64,5081	78,1222	107,0353	500,3359
	8.	69,5097	78,1291	107,5269	499,7116
	9.	65,5249	62,2166	108,0259	495,0052
	10.	65,5310	62,5118	116,4986	501,0458
Średnia		65,81725	68,71731	108,4206	499,3377
Odchylenie standardowe		1,970343	8,046801	3,06108	7,20961

Tabela 6. Wyniki pomiarów czasu trwania wczytywania obrazu bez zastosowania filtru na platformie Android

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	311,595	595,477	1077,081	1963,799
	2.	347,015	582,990	1047,758	1952,180
	3.	329,200	565,402	1071,214	1931,721
	4.	331,272	550,589	1004,119	1994,243
	5.	324,357	581,034	1031,175	1872,798
	6.	302,834	600,055	1062,640	1917,439
	7.	351,377	553,275	1016,760	1954,589
	8.	352,768	637,666	1013,993	1900,748
	9.	313,108	595,436	1056,865	1980,196
	10.	302,274	611,104	1016,176	1969,109
Średnia		326,580	587,3028	1039,778	1943,6822
Odchylenie standardowe		19,185677	26,7537568	26,57354	37,759064

Tabela 7. Wyniki pomiarów czasu trwania wczytywania obrazu bez zastosowania filtru na platformie Windows 10 Mobile

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	286,7113	382,8265	455,2751	721,0278
	2.	282,2281	356,5726	433,3450	714,3192
	3.	272,1987	316,7453	498,5671	697,8735
	4.	275,5580	321,0543	514,9757	734,9586
	5.	285,0366	358,8566	484,0400	666,5381
	6.	288,0256	334,5981	501,2007	655,8544
	7.	260,8198	338,8908	468,3557	673,7759
	8.	277,4814	354,3209	454,1504	690,5607
	9.	249,8218	349,9877	486,7857	675,2426
	10.	301,6631	305,8886	458,1208	642,8750
Średnia		277,9544	341,97414	475,48162	687,30258
Odchylenie standardowe		14,65442	23,151483	25,7039056	29,782769

Tabela 8. Wyniki pomiarów czasu trwania wczytywania obrazu z zastosowaniem filtru Invert Color na platformie Android

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	450,955	742,429	1311,640	2939,336
	2.	463,117	725,451	1391,905	2980,945
	3.	472,645	772,903	1361,115	2860,552
	4.	439,756	788,207	1308,342	2916,138
	5.	399,070	718,667	1364,266	2840,092
	6.	446,607	769,982	1324,268	2828,857
	7.	461,681	730,500	1330,405	2994,383
	8.	435,950	793,799	1323,011	2918,065
	9.	474,052	707,653	1396,121	2864,818
	10.	442,487	787,835	1387,202	2961,229
Średnia		448,632	753,7426	1349,828	2910,4415
Odchylenie standardowe		21,975134	32,332681	34,27878	59,345783

Tabela 9. Wyniki pomiarów czasu trwania wczytywania obrazu z zastosowaniem filtru Invert Color na platformie Windows 10 Mobile

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	875,9948	985,2309	1282,3005	2145,4621
	2.	939,9395	1029,6358	1201,2254	2028,7621
	3.	872,5265	1060,9050	1276,1367	1977,8877
	4.	924,6700	1031,2712	1216,4592	2039,8187
	5.	915,2091	981,3049	1241,5329	2134,4103
	6.	851,2652	972,7751	1208,9054	2076,4461
	7.	907,1745	959,2436	1265,7399	2068,1005
	8.	911,0692	1002,1670	1203,6973	2096,6950
	9.	859,9536	1038,2657	1196,6281	2125,6613
	10.	884,5239	988,7826	1199,5672	2079,6493
Średnia		894,2326	1004,9582	1229,21926	2077,2893
Odchylenie standardowe		29,48787	33,140198	34,085145	52,022536

Tabela 10. Wyniki pomiarów czasu trwania operacji dodawania, modyfikowania oraz usuwania elementu listy na platformie Android

		Dodawanie [ms]	Modyfikowanie [ms]	Usuwanie [ms]
Pomiar	1.	115,291	91,112	2,992
	2.	98,114	100,127	3,224
	3.	97,016	89,197	3,206
	4.	103,978	85,640	3,192
	5.	98,504	92,716	3,119
	6.	87,124	118,434	3,137
	7.	81,205	96,558	3,112
	8.	102,348	107,599	3,247
	9.	79,643	87,968	3,347
	10.	96,291	111,527	3,164
Średnia		95,9514	98,0878	3,174
Odchylenie standardowe		10,817411	11,08283921	0,094615

Tabela 11. Wyniki pomiarów czasu trwania operacji dodawania, modyfikowania oraz usuwania elementu listy na platformie Windows 10 Mobile

		Dodawanie [ms]	Modyfikowanie [ms]	Usuwanie [ms]
Pomiar	1.	213,2765	200,8310	15,6274
	2.	222,6895	203,1583	20,9389
	3.	216,1872	267,1688	1,0011
	4.	238,1435	224,6395	4,0031
	5.	200,4493	215,9724	6,0340
	6.	248,7288	249,7972	4,0022
	7.	247,1553	231,7920	1,0039
	8.	247,4104	233,3013	4,0022
	9.	259,4967	235,9488	15,6515
	10.	232,0573	232,0764	4,0027
Średnia		232,55945	229,4686	7,6267
Odchylenie standardowe		18,916328	20,05894	7,0589414

Tabela 12. Wyniki pomiarów czasu potrzebnego do uzyskania, za pomocą GPS-u, współrzędnych geograficznych na platformach Android oraz Windows 10 Mobile

		Android [ms]	Windows 10 Mobile [ms]
Pomiar	1.	24763,093	26981,7056
	2.	4996,948	20502,6369
	3.	14102,487	8094,5966
	4.	8092,174	41586,0219
	5.	10246,351	16941,7255
	6.	8127,505	10565,7142
	7.	7136,173	7482,8751
	8.	10184,288	42251,9974
	9.	9116,895	12898,4021
	10.	16089,681	4605,6059
Średnia		11285,56	19191,12812
Odchylenie standardowe		5740,5913	13694,10964

6. Wnioski

Tworzenie jednej aplikacji na wiele platform z wykorzystaniem rozwiązań wieloplatformowych jest efektywne pod wieloma względami.

Przede wszystkim nie jest wymagana znajomość różnych języków programowania. Wymagana jest jedynie znajomość języka C#, ponieważ Xamarin jest platformą opartą o platformę .Net oraz architekturę Windows Runtime.

Tworząc aplikację udało się uzyskać część wspólną kodu na poziomie 85% linii fizycznych i 82% linii logicznych. Pozwoliło to znacząco skrócić czas programowania w porównaniu do pisania oddzielnych aplikacji na każdą z platform.

Aplikacja ma zaimplementowane wiele funkcji i korzysta z dużej liczby bibliotek, więc jej rozmiar wynoszący 32 MB na platformie Android nie jest zbytnio wygórowany.

Operacja zapisu wykonywana jest szybciej na urządzeniu z systemem Android, natomiast w przypadku odczytu to Windows 10 Mobile jest szybszy.

Na czas wczytywania obrazu wpływ ma waga pliku oraz zastosowanie filtru. Zwiększenie rozmiaru pliku powoduje wydłużenie czasu wczytywania. Podobny efekt daje użycie filtru. W większości przypadków lepsze wyniki osiągnął Windows 10 Mobile. Może to świadczyć o tym, że platforma UWP, która powstała w oparciu o architekturę Windows Runtime, radzi sobie lepiej z aplikacją stworzoną w oparciu o tę samą architekturę, niż platforma, na której aplikacja uruchamiana jest w połączeniu z Javą, tak jak w przypadku Androida.

Średnie czasy dodawania i modyfikowania elementów listy są do siebie podobne i ok. 30 razy wyższe od średniego czasu usuwania elementu. Sytuacja taka ma miejsce w przypadku obu testowanych platform. Dodatkowo, średnie wartości dla systemu Android są ok. 2,3-2,4 razy niższe niż w przypadku Windows 10 Mobile.

W badaniu czasu uzyskiwania współrzędnych z pomocą GPS-u otrzymano zróżnicowane wyniki. Wystąpiły w nim wartości w zakresie od kilku do kilkudziesięciu sekund. Choć średnia wskazuje na korzyść Androida, należy pamiętać, że na pracę systemu nawigacji wpływać może wiele czynników, wewnętrznych oraz zewnętrznych.

Pomimo dłuższych czasów wykonywania niektórych operacji, można uznać, że poziom wydajności aplikacji stoi na zadowalającym poziomie. Ponadto duża część kodu jest wspólna dla wszystkich platform docelowych. W związku z tym, należy stwierdzić, że postawiona na początku hipoteza badawcza została potwierdzona.

Literatura

[1] Martinez M., Lecomte S., Towards the quality improvement of cross-platform mobile applications, 2017.

[2] Gerasimov V., Bilovol S., Ivanova K., Comparative Analysis Between Xamarin and Phonegap for .Net, System technologies, 2015, vol. 96.

[3] Radi A., Evaluation of Xamarin Forms for Multi-Platform Mobile Application Development, Technical Library, 2016, paper 249.

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

[5] Peppers J., Xamarin 4.x Cross-Platform Application Development. Third Edition, Packt Publishing, 2016.

[6] Build a Native Android UI & iOS UI with Xamarin.Forms, https://www.xamarin.com/forms [22.11.2017].

[7] Tunalı V., Erdogan S., Comparison of Popular Cross-Platform Mobile Application Development Tools, 2. Ulusal Yönetim Bilişim Sistemleri Kongresi (YBS2015), Erzurum, 2015.

[8] Johnson P., Cross-platform UI Development with Xamarin.Forms, Packt Publishing, 2015.

[9] Serializing and Deserializing JSON, https://www.newtonsoft.com/json/help/html/SerializingJSON.htm [22.11.2017].

[10] Daniel Plaisted, PCL Storage, https://github.com/dsplaisted/PCLStorage [dostęp 22.11.2017].

[11] James Montemagno, Take & Pick Photos and Video Plugin for Xamarin and Windows, https://github.com/jamesmontemagno/MediaPlugin [22.11.2017].

[12] James Montemagno, Checking Current Location, https://jamesmontemagno.github.io/GeolocatorPlugin/CurrentLocation.html [dostęp 22.11.2017].

[13] Map – Xamarin, https://developer.xamarin.com/guides/xamarin-forms/user-interface/map/ [22.11.2017].

[14] Google Maps Geocoding API, https://developers.google.com/maps/documentation/geocoding/intro [22.11.2017].

[15] Daniel Luberda, FFImageLoading - Xamarin.Forms API, https://github.com/luberda-molinet/FFImageLoading/wiki/Xamarin.Forms-API [22.11.2017].

[16] Sencer Sultanoğlu, Software Size Estimating, http://yunus.hun.edu.tr/~sencer/size.html [22.11.2017].