JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 162-167
Wysłane: 2017-12-02
Przyjęte: 2017-12-09

Analiza wydajnościowa platformy Ionic 2

Robert Pyć*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono analizę wydajnościową platformy Ionic 2. Poddano analizie dwie aplikacje jedna napisaną w języku natywnym Android, czyli w Javie, druga napisano z pomocą frameworka Ionic 2. Do analizy frameworka wybrano następujące kryteria: czas renderowania różnych rodzajów multimediów. płynność działania, zużycie zasobów oraz działanie aplikacji przy obciążonym systemie. Przeprowadzone badania dowiodły iż framework działa znacznie wolniej od aplikacji napisanej w natywnym framework'u Androida.

Słowa kluczowe: aplikacja hybrydowa, Ionic, Ionic 2, Android, Wydajność frameworka

*Autor do korespondencji.

Adres e-mail: robert.pyc@pollub.edu.pl

Efficiency analysis of the Ionic 2 platform

Robert Pyć*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The paper presents efficiency analysis of the Ionic 2 platform. Two test applications were analysed, one written in the native Android language, Java, and the second written using the Ionic 2 framework. The following criteria were selected for the framework analysis: rendering time of various media types, fluidity, resource consumption and application working while system load. The study has shown that the framework works much slower than an application written in native Android language.

Keywords: hybrid application; Ionic; Ionic 2 Android; Efficiency of framework

*Corresponding author.

E-mail address: robert.pyc@pollub.edu.pl

1. Wstęp

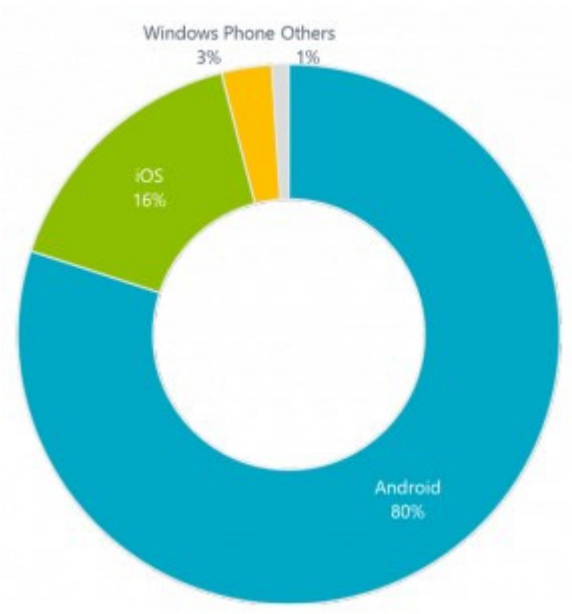
W ostatnich latach liczba użytkowników urządzeń mobilnych znacząco wzrosła w porównaniu do wcześniejszych. lat. Według raportu THE RADICATI GROUP, INC. w 2018 roku liczba użytkowników urządzeń mobilnych sięgnie 6.2 mld, co stanowi około 84% ludzkości. Przewiduje się także, że liczba urządzeń mobilnych przekroczy 12 mld[2][3].

	2014	2015	2016	2017	2018
Worldwide Mobile Users (M)	5,674	5,808	5,945	6,085	6,228
Total Mobile Devices* (M)	7,733	8,627	9,628	10,825	12,165
Mobile Devices Per Business User	1.36	1.49	1.62	1.78	1.95

Rys. 1. Liczba urządzeń mobilnych oraz użytkowników w latach 2014-2018 [2]

Tak dynamicznie wzrastająca liczba użytkowników otworzyła nowe możliwości dla programistów. W 2016 roku liczba aplikacji w Google Play sięgnęła 1,6 mln[4]. Obecnie na rynku urządzeń mobilnych dominują trzy systemy: iOS, Android oraz Windows Phone. Udział w rynku urządzeń mobilnych w roku 2016 prezentuje rysunek 2 [5].

Każdy system cechuje się innym językiem programowania, dla przykładu dla systemu Android jest to język Java, dla Windows Phone C# a dla systemu iOS Objective C/Swift. Istnieją także aplikacje hybrydowe, które dzięki zastosowaniu języka HTML, JavaScript oraz CSS, umożliwiają uruchamianie na dowolnej platformie mobilnej[7].



Rys. 2. Udział w rynku urządzeń mobilnych w roku 2016[6].

Aplikacje hybrydowe stają się obiecującym rozwiązaniem do obsługi wielu platform mobilnych. Dostarczają zarówno specyficzne narzędzia dla platformy oraz interakcje użytkownika za pomocą kodu JavaScript [8]. Aplikacje hybrydowe pomagają programistom tworzyć wiele

programów dla różnych platform bez większego nakładu pracy. Jednym z frameworków, które umożliwiają takie programowanie jest Ionic 2. Zapewnia on użytkownikom wszystkie komponenty, narzędzia i funkcjonalności używane w rodzimym rozwoju aplikacji mobilnych[9][10].

1.1. Cel i obszar badań

Celem badań jest sprawdzenie wydajności framework'a Ionic 2, poprzez stworzenie dwóch aplikacji o tych samych funkcjonalnościach. Pierwsza z nich została napisana w języku natywnym Androida - Java, przy użyciu środowiska Android Studio. Druga napisana została przy użyciu Ionic 2. W ten sposób będzie możliwe zbadanie jak framework radzi sobie z renderowaniem różnych rodzajów multimediów.

W celu dokładnego przeprowadzenia analizy określono następujące kryteria badawcze: zużycie zasobów badanego urządzenia, czas obsługi zlecenia, szybkość działania aplikacji, działanie aplikacji podczas obciążenia systemu przez inne oprogramowania pracujące w tym samym momencie na testowanym urządzeniu.

1.2. Hipotezy badawcze

W niniejszym artykule pod tytułem „Analiza wydajności framework'a Ionic 2" postawiono hipotezę: "Aplikacje stworzone za pomocą framework'a Ionic 2 są wydajniejsze niż aplikacje napisane przy użyciu natywnego framework'a." Zostanie to sprawdzone za pomocą wyżej wymienionych kryteriów.

2. Materiały i metody

Przyjęto, iż do przetestowania framework'a Ionic 2 najlepsza będzie aplikacja menadżera plików multimedialnych. Przyjęto iż będzie on odtwarzał następujące typy plików: obrazy o rozszerzeniach jpg oraz jpeg, animacje typu gif, krótkie filmy o rozszerzeniu mp4 i 3gp, jak również pliki tekstowe txt.

W odtworzenia obrazów dodano dodatkową funkcjonalność polegającą na zmianie jasności oraz kontrastu wybranego przez użytkownika obrazu.

Podobnie jak w przypadku obrazów do odtworzenia plików tekstowych dodano dodatkową funkcjonalność, wyszukiwanie wzorca w tekście.

3. Kryteria analizy

Poniżej skupiono się na dokładniejszym opisie wszystkich określonych wcześniej kryteriów służących sprawdzeniu wydajności framework'a Ionic 2.

Pierwszym badanym kryterium było zużycie zasobów. Zbadane zostało przy użyciu zewnętrznego oprogramowania. Pomogło ono zdobyć informacje o zużyciu procesora oraz pamięci RAM podczas działania aplikacji natywnego framework'a, jak i tej napisanej przy pomocy Ionic 2. Zużycie sprawdzano na czterech rodzajach multimediów: krótkich filmach, animacjach, dokumentach tekstowych w formacie txt, obrazach o różnych rozdzielczościach

Jako drugie kryterium przyjęto czas obsługi zlecenia. Kryterium to zostanie zbadane dzięki zaprogramowaniu w aplikacji funkcjonalności, która po wykonaniu zleconego zadania pokaże informacje z czasem realizacji w milisekundach. Zostanie to zbadane zgodnie z wcześniej przedstawionymi kryteriami, badając różne pliki o rożnych wielkościach oraz rozdzielczościach.

Kolejnym kryterium poddanym analizie została szybkość działania. W tym kryterium brano pod uwagę czas jaki jest potrzebny do uruchomienia aplikacji oraz czy poszczególne ekrany aplikacji wczytują się płynnie.

Jako ostatnie postanowiono zbadać działanie aplikacji przy równoczesnym obciążeniu systemu innymi procesami, oraz późniejszym ponownym przetestowaniu aplikacji pod kątem wszystkich powyższych kryteriów.

3.1. Wyszukiwanie wzorca w tekście

Dany test polega na wyszukaniu w dużej ilości tekstu konkretnego fragmentu tekstu, czyli wzorca. W folderze znajdują się pliki z tekstem o rozszerzeniu txt. Są one różnego rozmiaru od 200kB do 2 MB.

Fragment kodu przedstawiony poniżej (Rys. 3) przedstawia szukanie w tekście wzorca. Jeżeli zostanie on znaleziony, szukany fragment podświetla się na czerwono.

Przykład. 1. Fragment aktywności TextActivity odpowiedzialny za wyszukanie wzorca w aplikacji Androida

```
 if (fullText.contains(criteria)) {
                int indexOfCriteria = fullText.indexOf(criteria);
                int lineNumber =
tv.getLayout().getLineForOffset(indexOfCriteria);
                String highlighted = "<font
color='red'>"+criteria+"</font>";
                fullText = fullText.replace(criteria, highlighted);
                tv.setText(Html.fromHtml(fullText));
                textWrapper.scrollTo(0,
tv.getLayout().getLineTop(lineNumber));
            }else {
                  fullText = fullText.replace("<font
color='red'>", "");
                  fullText = fullText.replace("</font>", "");
                 tv.setText(fullText);
            }
```

3.2. Liczenie czasu wykonywania zadania

Została zaprogramowana funkcjonalność, dzięki której po zakończeniu wykonywania każdej czynności aplikacja wyświetla czas jaki został poświęcony na realizację. Poniższy fragment kodu (Rys. 4) przedstawia przykładową metodę odpowiedzialną za wyświetlenie czasu.

Przykład. 2: Przykładowy kod pokazujący czas realizacji zadania w aplikacji Androida

```
protected void onPostExecute(Bitmap bitmap) {
            super.onPostExecute(bitmap);
            if (bitmap != null) {
               ImageView myImage = (ImageView)
findViewById(R.id.imageView);
               myImage.setImageBitmap(bitmap);
            }
            long time = System.currentTimeMillis() -
startTime;
            Toast.makeText(PhotoActivity.this,"Czas ładowania
zdjęcia: "+time + "ms",Toast.LENGTH_LONG).show();
            progressbar.dismiss();
        }
```

3.3. Realizacja operacji na plikach graficznych

Analizę operacji na plikach graficznych postanowiono przeprowadzić poprzez zastosowanie obrazów o różnej rozdzielczości oraz użycie filtrów w postaci zmiany kontrastu oraz jasności wyświetlonego obrazu. Po każdej zmianie filtrów wyświetlany jest czas realizacji zdania.

W przypadku aplikacji androida funkcja ta została zrealizowana poprze metodę którą reprezentuje rysunek 5. Przyjmuje ona wartości jasności oraz kontrastu jako parametry a następnie poprzez wykorzystanie metody colormatrix w obiekcie Paint rysuje nowa bitmapę.

Przykład 3. Metoda zmiany jasności oraz kontrastu w aplikacji androida

```
public static Bitmap changeBitmapContrastBrightness(Bitmap
bmp, float contrast, float brightness,float saturation)
   {
      ColorMatrix cm = new ColorMatrix(new float[]
            {
                  contrast, 0, 0, 0, brightness,
                  0, contrast, 0, 0, brightness,
                  0, 0, contrast, 0, brightness,
                  0, 0, 0, 1, 0
            });
      Bitmap ret = Bitmap.createBitmap(bmp.getWidth(),
bmp.getHeight(), bmp.getConfig());

      Canvas canvas = new Canvas(ret);
      Paint paint = new Paint();
      paint.setColorFilter(new ColorMatrixColorFilter(cm));
      canvas.drawBitmap(bmp, 0, 0, paint);
      return ret;
   }
}
```

W przypadku aplikacji Ionic 2 zastosowano style CSS. Poprzez fragment kodu przedstawiony na rysunku 6 można w czasie rzeczywistym zmieniać jasność oraz kontrast wybranego obrazu.

Przykład 4. Funkcja w aplikacji Ionic 2 do zmiany jasności oraz kontrastu

```
<img id="myImage" src="{{ data }}" [ngStyle]="{'filter':
'brightness(' + brightness/100 + ') '+'contrast(' +
contrast/100 + ')'}"/>
```

3.4. Odtwarzanie animacji gif

Do prawidłowego odtworzenia animacji gif zainstalowano dodatek Glide, który wspomaga ich wyświetlanie oraz umożliwia ich poruszanie.

4. Badania

Badania zostały przeprowadzone na trzech urządzeniach mobilnych o różnej charakterystyce sprzętowej. Były to:

1) Sony Xperia M2
2) Xiaomi Redmi 4x
3) Xiaomi Mi 6

Na samym początku zbadano czas renderowania poszczególnych elementów multimedialnych, zwracając uwagę na zużycie procesora oraz pamięci RAM. Użycie zasobów urządzenia mobilnego zbadano z pomocą narzędzia “Tinycore”. Oprogramowanie to działając w tle zbiera informacje o aktualnym stanie użytkowania procesora oraz RAM-u a następnie wyświetla je w postaci paska

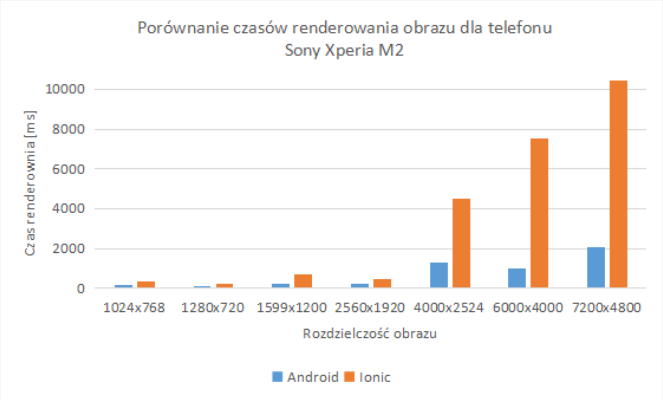
4.1. Galeria

Jako pierwszą postanowiono podda badaniu galeria obrazów. Poniższa tabela (Tabela 1) reprezentuje czas renderowania obrazów z galerii podany w milisekundach.

Tabela 1. Czas renderowania obrazów z galerii. Dane w ms

Rozdzielczość grafiki	Sony Xperia M2		Xiaomi Mi 6		Xiaomi Redmi 4x	
	Android	Ionic	Android	Ionic	Android	Ionic
1024x768	150	367	47	121	76	266
1280x720	114	193	45	69	84	154
1599x1200	235	672	68	141	131	389
2560x1920	210	485	87	147	137	427
4000x2524	1289	4517	250	1246	707	3683
6000x4000	1014	7541	220	1357	560	4861
7200x4800	2081	10456	455	1548	1125	5587

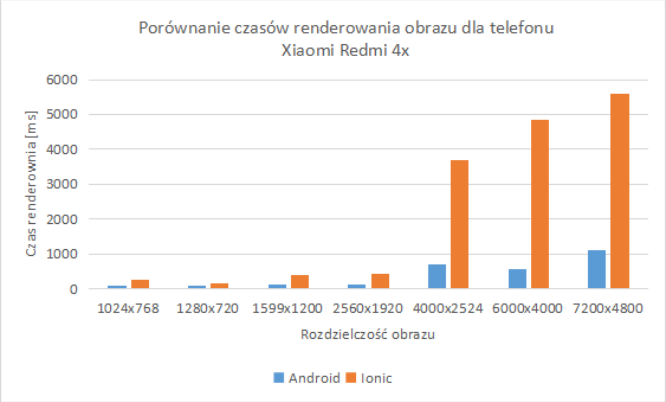
Poniższe wykresy prezentują wykresy słupkowe czasu renderowania obrazów z galerii dla telefonów Sony Xperia M2 (rysunek 3), Xiaomi Redmi 4x (rysunek 4), Xiaomi Mi 6 (rysunek 5). Oś X oznacza rozdzielczość prezentowaną w pikselach natomiast oś X czas renderowania grafiki wyrażony w milisekundach.



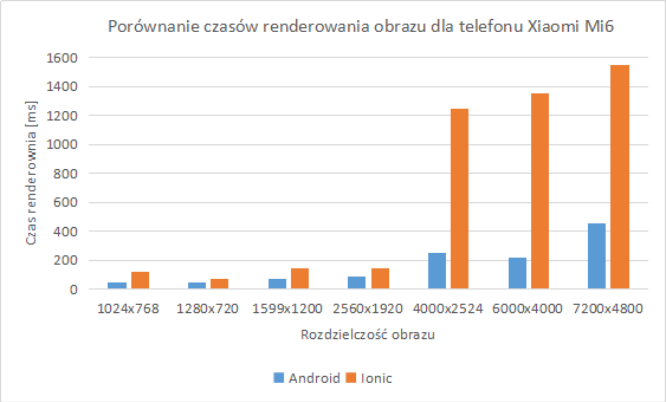
Rys 3. Porównanie czasów renderowania obrazu dla telefonu Sony Xpieria M2.

W przypadku telefonu marki Sony można było zauważyć znaczne zużycie procesora, zwiększało się o ok 30% przy uruchamianiu obrazów szczególnie tych o większej rozdzielczości. Przy badaniu zużycia pamięci Ram nie pojawiły się znaczące zmiany.

Przy badaniu wydajności aplikacji na telefonach marki Xiaomi nie zauważano wzrost zużycia procesora w przypadku otwierania obrazów o ok 10-15%, natomiast nie odnotowano zmiany w użyciu pamięci RAM.



Rys 4. Porównanie czasów renderowania obrazu dla telefonu Xiaomi Redmi 4x



Rys 5. Porównanie czasów renderowania obrazu dla telefonu Xiaomi Mi6

Następnie skupiono się na wykorzystaniu funkcjonalności zmiany jasności oraz kontrastu. W przypadku aplikacji napisanej za pomocą frameworka Ionic 2, można zaobserwować zmiany wprowadzone przez filtry w czasie rzeczywistym. Zauważono także, iż użycie procesora w przypadku dynamicznej zmiany filtrów rośnie na wszystkich badanych urządzeniach, Xperia około 20%, Redmi 4x w przybliżeniu 50%, natomiast na Mi6 około 10%.

W oprogramowaniu napisanym w natywnym framework'u Androida, czas potrzebny na wczytanie pliku graficznego z nałożonym filtrem nie różni się znacznie od czasu potrzebnego do wczytania oryginalnego obrazu. Zostało to potwierdzone na wszystkich urządzeniach mobilnych, co przedstawia tabela 2.

Tabela 2. Czas renderowania obrazów z galerii w ms wraz z czasem renderowania filtrów

Rozdzielczość obrazu	**Sony Xperia M2**		**Xiaomi Mi 6**		**Xiaomi Redmi 4x**	
	Android	**Filtr**	**Android**	**Filtr**	**Android**	**Filtr**
1024x768	150	153	47	45	76	75
1280x720	114	107	45	50	84	90
1599x1200	235	267	68	74	131	145
2560x1920	210	245	87	91	137	140
4000x2524	1289	1157	250	264	707	763
6000x4000	1014	1057	220	206	560	497
7200x4800	2081	2154	455	502	1125	1245

Następnie powtórzono powyższe badania dla systemu obciążonego innymi aplikacjami działającymi w tle. W przypadku urządzenia Xiaomi Mi 6, nie udało się obciążyć pamięci RAM oraz procesora w stopniu który powodowałby zmiany w uzyskanych czasach renderowania plików graficznych. Przy smartfonie Xiaomi Redmi 4x zwiększono zużycie procesora o 50% oraz zużycie RAM-u do około 80%. Jednak, co zaskakujące, nie udało się zaobserwować wydłużenia czasu renderowania obrazów w żadnym z przypadków.

4.2. Animacje gif

Następnie skupiono się na przeanalizowaniu animacji gif. Testy pomogły uzyskać informacja na temat czasu jaki aplikacja potrzebowała na uruchomienie danego pliku.

Jako pierwszą sprawdzono aplikacje Ionic 2. Stwierdzono iż w przypadku urządzenia Sony Xperia M2 oraz animacji o wymiarach 240x320 framework potrzebuje średnio 189 ms natomiast animacje o mniejszych wymiarach około 97 ms. Następnie proces powtórzono na urządzeniu Xiaomi Mi6,w tym przypadku większe animacje wczytywać się ok 98 ms natomiast mniejsze potrzebowały 53 ms. Jako ostatnie urządzenie sprawdzono Xiaomi Redmi 4x. W tym przypadku czasy ładowania wynosiły niemalże tyle samo co w przypadku urządzenia marki Sony.

Aplikacja napisana w natywnym framework'u Androida potrzebowała jedynie około 2 ms na otworzenie dowolnej animacji w przypadku wszystkich urządzeń

Nie zauważono stałych wzrostów zużycia procesora oraz pamięci RAM. Wystąpiło jedynie tymczasowe podwyższenie zużycia procesora przy otwieraniu pliku.

Następnie powtórzono powyższe badania, ja w przypadku galerii, na obciążonych systemach. Jednakże nie wpłynęło to na wydłużenie czasu ładowania plików, czy też zwiększenie zużycia zasobów.

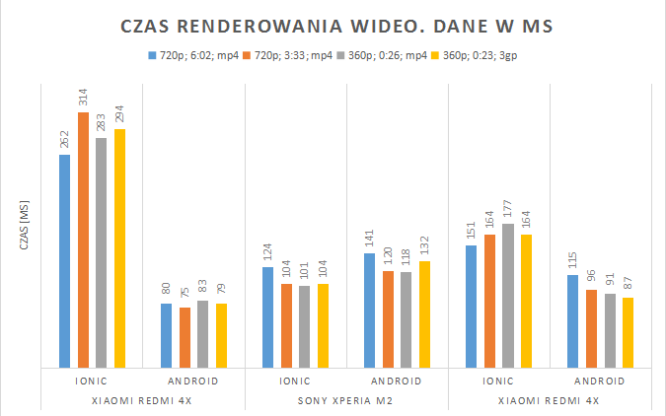
4.3. Wideo

Podobnie jak w przypadku galerii oraz animacji badanie plików wideo polegało na zmierzeniu czasu wczytywania poszczególnych filmów. Do badanie wybrano pliki z rozszerzeniem .mp4 oraz .3gp o jakościach 720p oraz 360p. Wyniki badania prezentuje poniższa tabela 3.

Tabela 3. Czas wczytywania wideo. Dane w ms oraz w minutach i sekundach

Jakość filmu	**Długość wideo [mm:ss]**	**Xiaomi Mi6**		**Sony Xperia M2**		**Xiaomi Redmi 4x**	
		Ionic	**Android**	**Ionic**	**Android**	**Ionic**	**Android**
720 p	6:02	262	80	124	141	151	115
720 p	3:33	314	75	104	120	164	96
360 p	0:26	283	83	101	118	177	91
360 p	0:23	294	79	104	132	164	87

Poniższy wykres (rysunek 6) prezentuje czas jaki jest potrzebny do wczytania filmu uwzględniając długość filmu jego rozszerzenie jak i język aplikacji na której jest on uruchamiany.



Rys 6. Czas renderowania wideo. Dane w ms, długość filmu podana w formacie minuty:sekundy.

Jak w przypadku poprzednich kryteriów tak i w tym teście badano wpływ obciążenia systemu na płynność wczytywania wideo. Różnice były minimalne, mieściły się w granicach 20 ms.

Analiza zużycia zasobów w trakcie uruchamiania oraz trwania wideo wykazała nieznaczny stały wzrost zużycia procesora, wynosił on mniej niż 5% w przypadku badanych urządzeń. Nie nie zaobserwowano zmian w zużyciu pamięci RAM.

4.4. Tekst

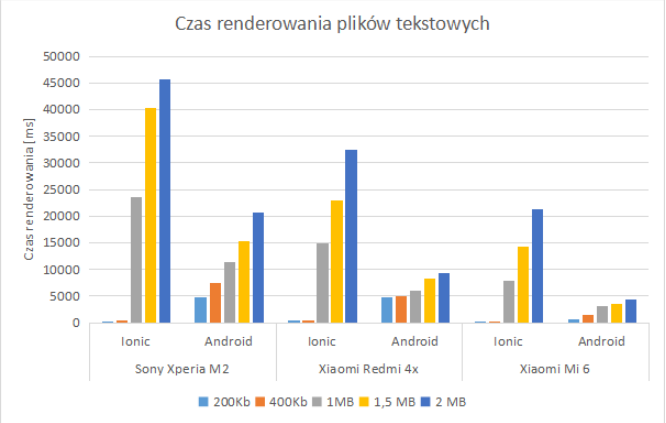
W badaniach na plikach tekstowych brano pod uwagę czas wczytania poszczególnych plików oraz czas potrzebny do wyszukania podanego wzorca w wybranym tekście Poniższa tabela 4 przedstawia długość oczekiwania na wykonanie zadania.

Tabela 4. Czasy wczytywania plików tekstowych. Dane w ms

Rozmiar pliku	Sony Xperia M2		Xiaomi Redmi 4x		Xiaomi Mi 6	
	Ionic	Android	Ionic	Android	Ionic	Android
200Kb	284	4862	192	4812	74	732
400Kb	390	7462	362	4951	100	1495
1MB	23591	11468	14897	6045	7941	3154
1,5 MB	40245	15286	23007	8253	14286	3549
2 MB	45721	20648	32468	9365	21279	4281

Rysunek 7 ilustruje porównanie czasów wczytywania plików tekstowych, uwzględniając rozmiar, dla poszczególnych urządzeń mobilnych.

Następnie poddano badaniu opcje wyszukiwania wzorca w tekście. W przypadku aplikacji Ionic 2 jedynie na urządzeniu Xiaomi Mi6 było możliwe płynne wyszukanie. Na pozostałych urządzeniach, przy próbie wpisania szukanego wyrazu, oprogramowanie nie reagowało. Czasy realizacji zadania przedstawia tabela 5



Rys 7. Porównanie czasów renderowania plików tekstowych

Tabela 5. Czas wyszukania wzorca w tekście. Dane w ms

Rozmiar pliku	Xiaomi Mi 6		Sony Xperia M2		Xiaomi Redmi 4x	
	Ionic	Android	Ionic	Android	Ionic	Android
200Kb	5412	425		3052		1608
400Kb	5971	970		6815		4703
1MB	6874	1246		8561		7539
1,5 MB	7932	1701		10241		8824
2 MB	10548	2111		15262		11093

Przy odczycie plików tekstowych zauważono znaczne zwiększenie zużycia procesora, trwające ok 5-10 sekund po zakończeniu realizacji zadania. W przypadku urządzenia Sony zaobserwowano zużycie procesora wynoszące 100%. Jeśli chodzi o pozostałe telefony wzrost był zauważalny, lecz znacznie mniejszy niż przy smartfonie marki Sony, gdyż wynosił on 25% dla urządzenia Xiaomi Redmi 4x oraz około 10% dla urządzenia Xiaomi Mi6.

4.5. Płynność działania

Jako ostatnie zbadano płynność działania. Już przy uruchamianiu obu aplikacji zauważono iż napisana w języku Androida uruchamia się znacznie szybciej niż Ionic 2.

W przypadku galerii, wczytywanie obrazów o mniejszej rozdzielczości odbywa się płynnie natomiast przy większych rozdzielczościach trzeba poczekać na wczytanie.

Pliki wideo oraz animacje gif w obu aplikacjach działają płynnie oraz nie można dostrzec żadnych opóźnień.

Zarówno aplikacja Androida jak i Ionic'a miała pewnie problemy z plikami tekstowymi. Czas oczekiwania na wczytanie całego pliku jest dłuższy przy większych plikach sięga nawet do 45 sekund w przypadku urządzenia marki Sony.

Wyszukiwanie w aplikacji Ionic 2, jak już było wspomniane wcześniej, udało się zrealizować jednie na urządzeniu o najlepszych parametrach sprzętowych, czyli Xiaomi Mi6. W przypadku reszty urządzeń czynność ta była niemożliwa ze względu na zawieszanie się aplikacji. W przypadku aplikacji napisanej w języku Androida, wyszukanie było możliwe, jednak wymagało ono chwilowego oczekiwania.

5. Wnioski

W niniejszym artykule przyjęto hipotezę iż aplikacje napisane z użyciem framework'a Ionic 2 są wydajniejsze od aplikacji napisanych w natywnym framework'u Androida. Dużym zaskoczeniem były wyniki badań aplikacji napisanej za pomocą framework'a Ionic 2. Spodziewano się iż okaże się ona lepszym rozwiązaniem pod względem wydajnościowym. Jednak testy jednoznacznie obalają postawioną hipotezę.

Ionic 2 może okazać się dobrym rozwiązaniem do szybkiego tworzenia aplikacji typowo biznesowych dzięki temu, że korzysta z narzędzia Angular 2. Największą, a zarazem decydującą dla badań, zaletą pisania programu przy użyciu języków natywnych platform mobilnych, jest homogeniczność rozwiązań. Tworzony projekt jest przeznaczony tylko dla jednej platformy.

W przypadku obciążenia systemu, przez inne aplikacje, spodziewano się odnotować znaczny wzrost czasu otwierania poszczególnych multimediów, jednak wyniki badań wybranych kryteriów nie potwierdziły tego.

Literatura

[1] A. Tonini, L. Fischer, J Carlos Balzano de Mattos, L Brisolara Analysis and Evaluation of the Android Best Practices Impact on the Efficiency of Mobile Applications,

[2] https://www.radicati.com/wp/wpcontent/uploads/2014/01/Mobile-Statistics-Report-2014-2018-Executive-Summary.pdf [22.11.2017]

[3] G. Shrivastava, P. Kumar, Privacy Analysis of Android Applications: State-of-art and Literary Assessment.

[4] https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/[21.11.2017]

[5] https://www.computerworld.pl/news/Najpopularniejsze-platformy-mobilne-wykorzystywane-w-biznesie,393299.html [17.11.2017]

[6] X. Chen; Z. Zong, Android App Energy Efficiency: The Impact of Language, Runtime, Compiler, and Implementation.

[7] S. Helal,R. Bose W. Li, Mobile Platforms and Development Environments Synthesis Lectures on Mobile and Pervasive Computing.

[8] A. Gupta, A. Gaffar H, Hybrid Application Development using Ionic Framework & AngularJS.

[9] S. Lee; J. Dolby; S. Ryu, HybriDroid: Static analysis framework for Android hybrid applications.

[10] C. Griffith, Mobile App Development with Ionic 2: Cross-Platform Apps with Ionic, Angular .& Cordova , O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, 2017-04-07.