JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 11 (2019) 119-124
Wysłane: 2019-01-14
Przyjęte: 2019-01-22

Analiza różnic pomiędzy szkieletami aplikacji natywnych i wieloplatformowych

Kinga Aleksandra Łobejko*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Głównym celem niniejszego artykułu jest analiza różnic pomiędzy szkieletami aplikacjami natywnych i wieloplatformowych. Położono nacisk na przedstawienie technologii, które są obecnie wykorzystywane, ich rozwój i wykorzystanie na rynku. Omówiono wybrane technologie, które umożliwiają programowanie mobilne z podziałem na języki natywne i języki wysokiego poziomu. Przedstawiono różnice pomiędzy procesami wytwarzania aplikacji mobilnych.

Słowa kluczowe: aplikacje mobilne; wieloplatformowość; języki natywne; języki wyższego rzędu

*Autor do korespondencji.

Adres e-mail: kinga.lobejko@pollub.edu.pl

Analysis of the differences between frameworks of native applications and cross-platform

Kinga Aleksandra Łobejko*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The main purpose of the article is an analysis of differences between frameworks of native applications and cross-platform. The emphasis was put on presenting technologies that are currently used, their development and the situation on the market. The work includes a discussion of selected technologies used in mobile programming with division into native and high level languages. The differences between the processes of manufacturing mobile applications were presented.

Keywords: mobile application; cross-platform; native language; high-level programming

*Corresponding author.

E-mail address: kinga.lobejko@pollub.edu.pl

1. Wstęp

Nowe technologie są obecne na całym świecie. Większość ludzi ma dostęp do komputerów, smartfonów, tabletów i innych urządzeń wymagających obsługi programowej. Wraz z postępem technologicznym powstają też nowe rozwiązania i języki programowania.

Temat wieloplatformowości jest tematem coraz częściej pojawiającym się w artykułach naukowych. Wieloplatformowość jest wyzwaniem na dzisiejszym rynku [1]. Wiodącymi platformami, jakie są rozważane przy tworzeniu aplikacji, są iOS i Android. Nie tylko różnorodność pod względem systemów operacyjnych jest wyzwaniem. Dodatkowym aspektem, który twórcy oprogramowania muszą wziąć pod uwagę, jest programowanie na różne rodzaje urządzeń, między innymi telefony komórkowe, tablety, telewizory, komputery pokładowe, piloty czy urządzenia ubieralne. Obserwacje rynku [1] pozwalają na wyciągnięcie wniosków, iż Internet rzeczy ma coraz większy udział w życiu użytkowników. Twórcy oprogramowania nie mogą zatem pozostać bierni. Muszą brać pod uwagę przekształcający się rynek zbytu oprogramowania. Programiści w celu zredukowania kosztów wytwarzania oprogramowania korzystają z wieloplatformowych zintegrowanych środowisk programistycznych [2]. Ważnym czynnikiem jest możliwość wykorzystania wiedzy programistów aplikacji webowych do tworzenia aplikacji mobilnych [3]. Rozważane są jednak również minusy tworzenia hybrydowych aplikacji. Głównymi argumentami przeciw rozwiązaniom wieloplatformowym są ograniczenia dostępu do funkcjonalności sprzętowych, wariacje w kwestii doświadczeń użytkowników czy spadek wydajności [3, 4]. Mówiąc o hybrydowych aplikacjach mobilnych można dostrzec pewne trendy. W Google Play, w którym znajdują się aplikacje na platformę Android można dostrzec wzrost liczby aplikacji hybrydowych, szczególnie w takich kategoriach jak finanse, medycyna, transport, biznes, styl życia czy społeczności [3]. Najmniej aplikacji hybrydowych znajduje się w kategoriach: fotografia, muzyka i dźwięk, narzędzia, gry, personalizacja. Świadczy to o tym, że aplikacje wieloplatformowe zyskały popularność w przypadku aplikacji, które korzystają z dużych zasobów bazodanowych i wchodzą w proste interakcje z nimi. Natomiast gdy wymagane jest sięganie do zasobów sprzętowych programiści stawiają na natywne aplikacje [3].

Obecnie badania naukowe nie dotyczą już tylko porównania wytwarzania oprogramowania w językach natywnych i językach wyższego rzędu, a sięgają one porównań na szczeblu środowisk pozwalających tworzyć oprogramowanie na wiele platform [2]. Artykuły naukowe poruszają również kwestię rozwoju technologii wieloplatformowych. Pomimo niewątpliwych zalet platform Xamarin czy Ionic, fala krytyki jest równie znacząca [5].

Negatywne zarzuty dotyczą między innymi odczuć użytkowników względem rozwiązań natywnych.

2. Języki wykorzystywane podczas procesu wytwarzania aplikacji mobilnych

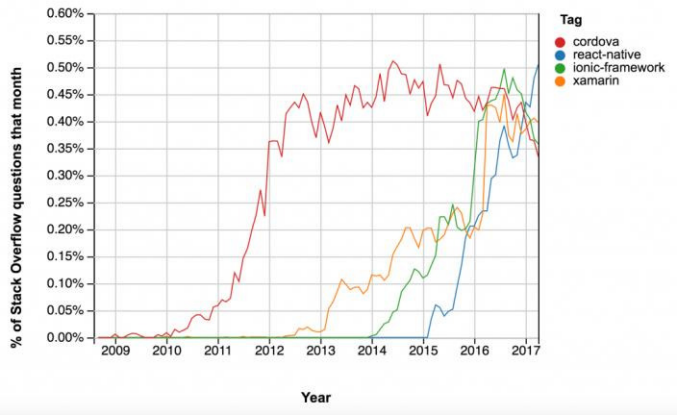
2.1. Języki natywne

Aplikacje natywne tworzone są na konkretną platformę. Rozwiązanie takie posiada szereg zalet, do których należą m.in.: szybkość działania aplikacji oraz posiadanie bezpośredniego dostępu do komponentów urządzanie mobilnego (np. akcelerometr, czujnik zbliżeniowy, aparat fotograficzny) [6]. Wadą aplikacji natywnych jest konieczność tworzenia osobnych wersji dla każdej platformy, co przekłada się na wydłużenie procesu wytwarzania oprogramowania.

Każda z platform posiada dedykowany język (bądź języki) programowania. W przypadku platformy iOS są to Objective-C oraz Swift, a dla platformy Android – Java i Kotlin. Z kolei językiem dedykowanym dla platformy Windows Phone jest C#. W niniejszym podrozdziale zostaną przedstawione języki natywne dla każdej z tych trzech platform.

2.2. Języki wysokiego poziomu

Alternatywą dla wytwarzania aplikacji mobilnych za pomocą języków natywnych są aplikacje hybrydowe. Są one tworzone z wykorzystaniem języków wysokiego poziomu (szkieletów).



Rys 1. Popularność szkieletów hybrydowych w pytaniach na portalu StackOverflow [7]

Na rys 1. zaprezentowano statystyki dotyczące częstości zapytań związanych z danym szkieletem do tworzenia aplikacji hybrydowych. Analizując powyższy wykres można zaobserwować kolejność powstawania kolejnych rozwiązań. Najstarszym szkieletem jest Apache Cordova (powstały w 2009 roku jako PhoneGap). Częstość zapytań o to rozwiązanie na portalu StackOverflow systematycznie rosła – wzrost ten zatrzymał się około roku 2012, gdy na rynku pojawiła się alternatywa – Xamarin. Trzecim rozwiązaniem jest IONIC, który stał się tematem pytań użytkowników w 2014 roku. Najmłodszym szkieletem w zestawieniu jest React Native. Analizując część powyższego wykresu dla roku 2017 można stwierdzić, że na początku 2017 roku użytkownicy najczęściej interesowali się najnowszym rozwiązaniem (jakim jest React Native). Drugie miejsce pod względem popularności zajmował Xamarin, trzecie z kolei – IONIC. Najmniej popularnym tematem z tej grupy było najstarsze rozwiązanie – Apache Cordova.

3. Rozróżnienie procesów wytwarzania aplikacji mobilnych

3.1. Natywna aplikacja mobilna

Natywna aplikacja mobilna jest to aplikacja napisana w językach domyślnie przeznaczonych na wybrane platformy mobilne, to znaczy dla iOS – Objective-C, Swift, dla Androida – Java, a dla WindowsPhone/Windows – C#. W przypadku wytwarzania oprogramowania z wykorzystaniem języków natywnych tworzony wspólny jest projekt aplikacji, jednak realizacji odbywa się w oddzielnych zespołach programistycznych, gdyż wytwarzany kod nie może być współdzielony.

Wśród popularnych, natywnych aplikacji mobilnych należy wymienić [8]: TechCrunch, The New York Times, Pokemon GO.

Rozwiązanie to posiada zarówno zalety, jak i wady. Niewątpliwie mocną stroną natywnych aplikacji mobilnych jest większa wydajność (w porównaniu z rozwiązaniami alternatywnymi) [9]. Aplikacje tworzone z wykorzystaniem języków natywnych (takich jak Java czy Objective-C) działają zazwyczaj szybciej, obciążając urządzenie w mniejszym stopniu. Kolejną zaletą aplikacji natywnych jest zapewnienie szybszego dostępu do danych – jest to bardzo istotny czynnik z punktu widzenia użytkownika produktu [10].

Niewątpliwie mocną stroną tego podejścia jest uzyskanie pełnego dostępu do możliwości urządzenia, na którym działa aplikacja (obsługa funkcji wbudowanych, takich jak akcelerometr, GPS i kamera). Programiści tworzący aplikację w języku natywnym danej platformy mają dzięki temu większe możliwości w kwestii funkcjonalności tworzonego produktu.

Kolejną zaletą tego rozwiązania jest uzyskanie większej przejrzystości architektury kodu aplikacji, w związku z czym zarządzanie kodem staje się łatwiejsze. W aplikacjach natywnych zazwyczaj stosowane są nowoczesne języki (Kotlin, Swift), które są dynamicznie rozwijane. Ponieważ każda platforma mobilna posiada swoje własne standardy, dlatego też aplikacje natywne posiadają zazwyczaj lepiej dostosowany *user experience* (doświadczenie użytkownika). Tego rodzaju podejście w wytwarzaniu aplikacji mobilnych wiąże się też zazwyczaj z dostępnością lepszych narzędzi do testowania (a co za tym idzie – łatwiejszym wykrywaniem błędów, które pojawiają się w trakcie implementacji aplikacji) [9].

Natywne aplikacje mobilne posiadają także swoje słabe strony. Po pierwsze, działają tylko w jednym systemie operacyjnym. Aby dana aplikacja działała na wielu platformach, musi ona zostać napisana w różnych językach (w języku natywnym danej platformy). Taka sytuacja rodzi koszty – firma jest zmuszona zatrudnić większą liczbę programistów – oraz powoduje wydłużenie czasu tworzenia aplikacji. Dany programista specjalizuje się najczęściej w jednym języku natywnym, a więc wytworzenie różnych wersji aplikacji dla różnych platform mobilnych oznacza zaangażowanie wielu zespołów programistów. Dlatego też wiele firm decyduje się na wykonanie swoich aplikacji firmowych tylko na jedną, wybraną platformę.

3.2. Hybrydowa aplikacja mobilna

Hybryda – jak sama nazwa wskazuje – jest połączeniem ze sobą dwóch różnych rozwiązań. Definicja ta występuje także w aspekcie aplikacji mobilnych. Przy budowie hybrydowej aplikacji mobilnej wykorzystuje się technologie, służące do tworzenia aplikacji internetowych, jednocześnie mając dostęp do natywnych funkcjonalności smartfonów. Cały ten proces może zachodzić dzięki narzędziom, które kompilują języki wyższego rzędu na języki natywne. W języku potocznym można nazwać to tłumaczeniem języka zrozumiałego dla aplikacji internetowej na język, którym posługuje się dana aplikacja mobilna.

Wśród aplikacji mobilnych wytworzonych tą metodą należy wymienić [12]: Amazon App Store, Twitter, Apple App Store, Evernote, Instagram, Untappd, Uber, Gmail.

Rozwiązanie to umożliwia stworzenie jednej aplikacji, która będzie działała na różnych systemach. Hybrydowe aplikacje mobilne wiążą się z ponownym wykorzystywaniem kodu – część kodu jest współdzielona, a część jest tworzona z myślą o konkretnej platformie. Czynnik ten sprawia, że w kwestii szybkości wytwarzania aplikacji działającej na wielu platformach (wynikającej z objętości tworzonego kodu) aplikacje hybrydowe górują nad aplikacjami stworzonymi z użyciem języków natywnych (w których wytwarzanie aplikacji na różne systemy wiąże się z tworzeniem kodu w językach dedykowanych dla danej platformy, a więc brakiem możliwości ponownego wykorzystania kodu).

Można wyróżnić dwa główne sposoby tworzenia hybrydowych aplikacji mobilnych. Pierwszy z nich polega na wykorzystaniu języka JavaScript oraz odpowiednich szkieletów, które – za pomocą wtyczek – pozwolą na dostęp do natywnych funkcji. Takie rozwiązanie osłabia jednak wydajność aplikacji. Drugi sposób związany jest natomiast z użyciem platform, które zostały oparte o bardziej natywne komponenty (przykładami są: Xamarin oraz React Native). Platformy te korzystają ze współdzielonej warstwy widoku oraz natywnych komponentów. Ponieważ komponenty te tworzone są w językach dedykowanych dla danej platformy, skutkuje to uzyskaniem wyższej wydajności (w porównaniu ze sposobem pierwszym) [9].

Wśród zalet hybrydowych aplikacji mobilnych należy wymienić m.in. uzyskanie oszczędności zasobów – wytworzenie tego rodzaju aplikacji nie wymaga zaangażowania osobnych zespołów programistycznych dla każdej platformy, na której będzie działać aplikacja. Dzięki takiemu podejściu skraca się również czas wytworzenia aplikacji – budowanie interfejsu użytkownika z wykorzystaniem technologii internetowych nierzadko okazuje się szybsze i łatwiejsze.

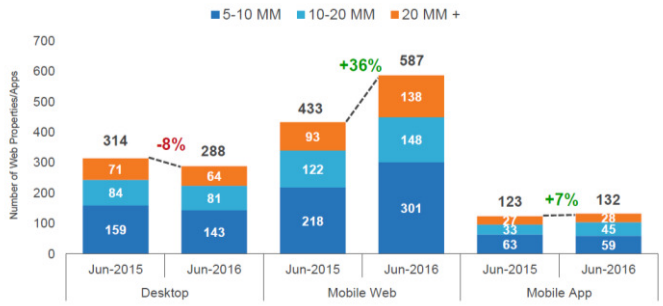
Hybrydowe aplikacje mobilne posiadają jednak także wady. Nie wszystkie szkielety stosowane w tym rozwiązaniu bezproblemowo radzą sobie z obsługą wbudowanych funkcji urządzenia (kwestia ta wygląda dużo lepiej w przypadku języków natywnych). Tego rodzaju aplikacje mogą również okazać się wolniejsze (ponieważ kod nie zawsze jest kompilowany do kodu napisanego z wykorzystaniem języka natywnego). W porównaniu z aplikacjami natywnymi, aplikacje hybrydowe posiadają zazwyczaj mniej dopasowany *user experience* (ponieważ nie są tworzone dla konkretnej platformy, tak więc konwencje mogą zostać uogólnione, przez co mogą odbiegać od konwencji danej platformy). Innymi słabszymi stronami tego rodzaju rozwiązania są częste zmiany w ekosystemie języka JavaScript oraz mniejszy potencjał do rozwoju aplikacji [9].

4. Progressive web app (PWA)

Progressive web app [13] (PWA) są to aplikacje internetowe, które uruchamiają się tak jak zwykłe strony internetowe [4] jednak wyglądają jak natywne aplikacje mobilne [13], co sprawia, że są przyjazne dla użytkownika. Jest to szczególnie atrakcyjne dla osób, które nie decydują się na wyszukiwanie aplikacji w sklepie, pobieranie jej i instalowanie ze świadomością, iż będzie użyta raz czy dwa razy. Statystycznie 49% użytkowników nie instaluje żadnej aplikacji w ciągu miesiąca [12]. Można zatem stwierdzić, że prawie połowa osób nie skorzysta z reklamujących się aplikacji. Według „The 2016 U.S. Mobile App Report" [12] użytkownicy korzystają z jednej ulubionej aplikacji przez większość czasu.

Jednocześnie znacznie chętniej użytkownik wejdzie na stronę internetową przez urządzenie mobilne. Dlatego też reklama stron i usług internetowych rozwinęła się adekwatnie do obecnych czasów i odpowiada na potrzeby konsumentów wykorzystując w tym celu PWA.

PWA są instalowane na głównym ekranie użytkownika bez potrzeby pobierania aplikacji ze sklepu z aplikacjami. Aplikacje mogą wykorzystywać powiadomienia jak aplikacje natywne. Dużą zaletą PWA jest szybkie załadowanie zawartości. Aplikacje uruchamiają się natychmiast – niezależnie od jakości połączenia internetowego. Na przestrzeni lat można dostrzec, iż wszystkie wymienione zalety zdecydowanie trafiają do użytkowników, co potwierdza rys. 2. Mobilne strony internetowe mają 4.5 razy więcej odwiedzin z aplikacji typu PWA niż z natywnych aplikacje mobilne, a także wzrost odwiedzin w przeciągu roku jest znacznie wyższy (o 29 punktów procentowych) [12].



Rys. 2. Liczba odwiedzin aplikacji internetowych, mobilnych aplikacji internetowych i aplikacji mobilnych oraz porównanie wartości w przeciągu roku [12]

Jedną z najważniejszych cech progresywnych aplikacji internetowych jest działanie w trybie offline. Jest to aspekt różniący PWA od zwykłych aplikacji internetowych, które w przypadku braku połączenia z siecią Internet nie wyświetlają użytkownikowi nawet części interfejsu bądź danych. Progresywne aplikacje internetowe w tej kwestii przypominają aplikacje natywne, które nawet w trybie offline są w stanie wyświetlić użytkownikowi treść (np. listę newsów pobranych podczas ostatniej wizyty w aplikacji). Ich działanie w trybie offline może sprowadzać się do wyświetlenia odpowiedniego komunikatu lub wyświetlenie danych pobranych w czasie, gdy aplikacja posiadała połączenie z siecią [14].

Elementem umożliwiającym wzbogacenie aplikacji o tego rodzaju działanie w trybie offline jest Service Worker API. Pozwala on programiście na określenie, które zasoby aplikacji zostaną zapisane do pamięci podręcznej, a które wymagają połączenia z siecią Internet. Dzięki takiej możliwości aplikacja jest w stanie wyświetlić użytkownikowi najważniejsze treści nawet w trybie offline.

Innym elementem, o który można uzupełnić aplikację dzięki Service Worker API są tzw. *push notifications*. Jest to rodzaj komunikatów, które mogą być wyświetlane nawet wtedy, gdy aplikacja nie jest otwarta [15]. Service Worker API pozwala także na zaimplementowanie w aplikacji *Background Sync* (API umożliwiające odraczanie działań, dopóki użytkownik nie uzyska stabilnego połączenia z siecią) [16].

Ważnym elementem progresywnej aplikacji internetowej jest plik manifest.json, przechowujący informacje dotyczące opisu aplikacji. Za pomocą tego pliku można zdefiniować m.in. lokalizacje ikon aplikacji, jej nazwę oraz kolory interfejsu przeglądarki i urządzenia mobilnego w momencie uruchamiania aplikacji. Ciekawą opcją jest możliwość ustawienia trybu „standalone” (w trybie tym interfejs przeglądarki zostaje ukryty – użytkownik odnosi wrażenie natywności uruchomionej aplikacji) [14].

Google opublikowało tzw. *Progressive Web App Checklist* [17], będący wykazem cech, które powinna spełniać aplikacja internetowa, żeby mogła być uznana za progresywną aplikację internetową. Zostały one podzielone na dwie grupy, które uznać można za: podstawowy poziom wymagań (*Baseline Progressive Web App Checklist*) oraz rozszerzony poziom wymagań (*Exemplary Progressive Web App Checklist*). Wśród wymagań poziomu podstawowego znajdują się m.in. następujące cechy:

- aplikacja internetowa jest serwowana przez HTTPS (zapewnione bezpieczeństwo danych),
- strony aplikacji są responsywne na tabletach i urządzeniach mobilnych,
- wszystkie adresy URL aplikacji ładują się w trybie offline (a więc aplikacja działa nawet wtedy, gdy urządzenie nie ma połączenia z siecią Internet),
- obecność metadanych pozwalających na dodanie skrótu aplikacji do ekranu startowego (umożliwia to plik manifest.json),
- aplikacja wczytuje się szybko nawet w przypadku wolniejszych połączeń (np. 3G),
- aplikacja internetowa działa poprawnie w różnych przeglądarkach (Chrome, Edge, Firefox, Safari, itp.),
- wszystkie przejścia i animacje są płynne, nie sprawiają one wrażenia, jakby aplikacja blokowała się z powodu wolnego (lub braku) połączenia z siecią Internet – aplikacja szybko reaguje na działania użytkownika, ekrany aplikacji zmianiają się sprawnie,
- każda strona aplikacji posiada swój adres URL.
- Rozszerzona lista wymagań obejmuje między innymi następujące wymagania:
- treść aplikacji (strony) jest indeksowana przez Google,
- strony używają History API,
- treść strony nie sprawia wrażenia „skakania” w trakcie ładowania,
- pola formularza nie są zasłaniane przez klawiaturę ekranową w momencie ich aktywowania (poprzez dotknięcie),
- aplikacja jest responsywna na telefonach, tabletach oraz urządzeniach desktopowych.

Wśród zalet progresywnych aplikacji internetowych należy wymienić fakt, że aplikacje tego typu uruchamiają się na każdej platformie (nie ma potrzeby tworzenia osobnych wersji aplikacji dla urządzenia mobilnego i urządzenia desktopowego – PWA będzie działać na każdej z tych platform). Drugą mocną stroną PWA jest łatwość aktualizacji (w przypadku aplikacji mobilnych - udostępnianych np. przez sklep z aplikacjami dostępny w systemie urządzenia – proces ten jest często wieloetapowy i długotrwały). Aktualizacja progresywnej aplikacji internetowej sprowadza się do wykonania aktualizacji plików na serwerze, a użytkownik podczas kolejnego uruchomienia tej aplikacji automatycznie otrzyma jej najnowszą wersję. Wybranie tego podejścia pozwoli również na redukcję kosztów, rozwiązując kwestię zatrudnienia programistów mobilnych. Ostatnią zaletą PWA jest wygoda korzystania z tego typu aplikacji – użytkownicy są w stanie uruchomić i korzystać z tego rodzaju aplikacji na każdej platformie [14].

Progresywne aplikacje internetowe posiadają również swoje słabsze strony. Ponieważ aplikacje te wykorzystują do swojego działania przeglądarki mobilne, tak więc niektóre funkcjonalności PWA mogą działać w różny sposób na różnych przeglądarkach (albo w ogóle nie funkcjonować).

Niektóre funkcjonalności telefonu nie są dostępne za pośrednictwem WEB API, co w pewien sposób ogranicza twórców progresywnych aplikacji internetowych [14].

5. Porównanie procesów wytwarzania aplikacji mobilnych

Aplikacje natywne posiadają przewagę nad aplikacjami hybrydowymi w kwestii wydajności. Podobnie przedstawia się kwestia dostępu do funkcji wbudowanych urządzenia mobilnego (czujniki, GPS, aparat) – dostęp do tych elementów za pośrednictwem języków natywnych jest łatwiejszy. Aplikacje natywne dysponują także lepszymi narzędziami służącymi do testowania, w związku z czym w przypadku tego rodzaju aplikacji wykrywanie błędów staje się łatwiejsze i szybsze.

Aplikacje hybrydowe posiada przewagę nad podejściem natywnym w wytwarzaniu aplikacji mobilnych w przypadku oszczędności zasobów. Do stworzenia aplikacji hybrydowej wymagane jest zaangażowanie jednego zespołu programistycznego. W przypadku aplikacji natywnych, nad wersjami aplikacji na różne platformy pracują zazwyczaj różne zespoły programistyczne.

Podejmując decyzję dotyczącą wyboru między podejściem natywnym i hybrydowym, należy rozważyć również inne kwestie. Pierwszą z nich jest kwestia budżetu – w tym przypadku wytworzenie aplikacji hybrydowej będzie wiązało się z mniejszymi wydatkami. Wynika to z braku konieczności zatrudnienia wielu zespołów programistycznych (w przypadku aplikacji natywnej zazwyczaj różne wersje aplikacji wymagają zaangażowania nowego zespołu programistycznego, skupiającego się na danej platformie). W przypadku aplikacji hybrydowych, duża część kodu jest współdzielona.

Kolejną kwestią jest wykorzystanie funkcji natywnych w aplikacji. W tym przypadku lepszą decyzją wydaje się być wybranie podejścia natywnego. Aplikacja wytworzona w języku natywnym danej platformy zazwyczaj będzie lepiej spełniała funkcje związane z dostępem do funkcji wbudowanych urządzenia – w przypadku aplikacji hybrydowej, niektóre, bardziej zaawansowane funkcjonalności mogą okazać się niedostępne lub trudniejsze w implementacji (w porównaniu z aplikacją natywną).

Rozważając sposób wytwarzania aplikacji mobilnej warto również wziąć pod uwagę doświadczenie użytkownika (*user experience*). W tym przypadku przewagą wykazują się aplikacje natywne. Stworzenie aplikacji na konkretną platformę wiąże się z przestrzeganiem pewnych konwencji ustalonych przez nią (np. w przypadku Google będzie to Material Design). Konwencje w aplikacjach hybrydowych są zazwyczaj bardziej uogólnione, ponieważ ta sama aplikacja będzie działać na różnych platformach.

Należy zastanowić się również nad tym, czy aplikacja może zostać wprowadzona na poszczególne platformy w sposób sekwencyjny. Jeśli odpowiedź na to pytanie jest twierdząca, wtedy aplikacja natywna może okazać się lepszym wyjściem. Aplikacja hybrydowa będzie miała przewagę nad podejściem natywnym w sytuacji, gdy produkt powinien być wprowadzony jednocześnie na wszystkie platformy – w przypadku aplikacji natywnej wymagałoby to jednoczesnego zatrudnienia osobnych zespołów programistycznych, którzy w tym samym czasie pracowaliby nad wersjami aplikacji na różne platformy. Sekwencyjne realizowanie kolejnych wersji aplikacji (w przypadku podejścia natywnego) ma również tę zaletę, że programiści realizujący kolejne wersje mają dostęp do wersji już stworzonych (a więc mogą korzystać z doświadczenia programistów, którzy już pracowali nad tą aplikacją).

W przypadku chęci szybkiego wejścia na rynek lepszą opcją jest aplikacja hybrydowa. Wynika to z szybszej implementacji aplikacji, która jest od razu dostępna na wielu platformach (bez konieczności tworzenia różnych wersji aplikacji, jak ma to miejsce w przypadku aplikacji natywnych).

Ostatnią kwestią jest rodzaj urządzeń, na których będzie działać aplikacja. Jeśli ma być ona uruchamiana nie tylko na telefonach, ale także na takich urządzeniach jak telewizory i zegarki, wtedy lepszym rozwiązaniem jest aplikacja natywna. Podejście to wynika z trudniejszego dostępu do funkcji natywnych w przypadku aplikacji hybrydowych.

6. Wieloplatformowość

Wieloplatformowość jest szerokim pojęciem, które może dotyczyć nie tylko oprogramowania, ale również języków programowania i systemów operacyjnych. W przypadku aplikacji, pojęcie to oznacza, że aplikacja działa na wielu (co najmniej dwóch) systemach operacyjnych i architekturze komputera.

Dzięki hybrydowym aplikacjom mobilnym można mieć do czynienia z wieloplatformowością. Kod jest pisany w jednym języku z wykorzystaniem jednego narzędzia, a stworzona aplikacja działa na wielu platformach. Zazwyczaj dotyczy to systemów iOS, Android i Windows. O wieloplatformowości można również mówić w przypadku progresywnych aplikacji internetowych (PWA). Aplikacje te nie wymagają konieczności tworzenia osobnych wersji aplikacji na różne platformy – od strony kodu są aplikacjami internetowymi, wzbogaconymi m.in. o działanie w trybie offline i tzw. *push notifications*.

7. Wnioski

W obecnych czasach urządzenia mobilne są wszechobecne. Zarówno w prywatnym życiu jak i zawodowym. Zwiększone zapotrzebowanie na urządzenia mobilne ma ogromny wpływ na rozwój nowych rozwiązań w zakresie oprogramowania. Urządzenia bez odpowiedniego oprogramowania nie są przydatne. Pierwotne rozwiązania jak języki natywne zostają powoli wypierane na rzecz nowych technologii. Powodem jest duże zróżnicowanie urządzeń wykorzystywanych przez konsumentów i nacisk na skrócenie czasu wytwarzania aplikacji przy ograniczeniu kosztów.

Literatura

[1] Majchrzak T. A., Rieger C.: Weighted Evaluation Framework for Cross-Platform App Development Approaches, EuroSymposium on Systems Analysis and Design, September 2016, DOI: 10.1007/978-3-319-46642-2_2 www.researchgate.net/publication/308495821 [09.12.2018]

[2] Goetz J., Li Y..: Evaluation of Cross-Platform Frameworks for Mobile Applications, September 2018, www.researchgate.net/publication/329488825_Evaluation_of_Cross-Platform_Frameworks_for_Mobile_Applications [09.12.2018]

[3] Malavolta I., Soru T., Terragni V.: End Users' Perception of Hybrid Mobile Apps in the Google Play Store, Mobile Services (MS), 2015 IEEE International Conference on, August 2015, [09.12.2018]

[4] E. Masi, G. Cantone, M. Mastrofini, G. Calavaro, P. Subiaco: Mobile apps development: A framework for technology decision making. In Mobile Computing, Applications, and Services, Springer, 2013.

[5] Biřrn-Hansen A., Majchrzak T. A., Grřnli1 Tor-Morten: Progressive Web Apps for the Unified Development of Mobile Applications, 2018, [09.12.2018]

[6] Rodzaje aplikacji mobilnych, www.biznesport.pl/blog/rodzaje-aplikacji-mobilnych/ [05.01.2019]

[7] Robinson D.: Exploring the State of Mobile Development with Stack Overflow Trends. www.stackoverflow.blog/2017/05/16/exploring-state-mobile-development-stack-overflow-trends/ [08.01.2019]

[8] Saccomani P.: Native Apps, Web Apps or Hybrid Apps? What's the Difference? www.mobiloud.com/blog/native-web-or-hybrid-apps/ [08.01.2019]

[9] Winkler M.: Native vs Hybrid – jaki rodzaj aplikacji mobilnej wybrać. www.appchance.pl/blog/native-vs-hybrid-jaki-rodzaj-aplikacji-mobilnej-wybrac/ [07.01.2019]

[10] Golan M.: Mobilna aplikacja internetowa czy natywna? A może hybryda? www.golan.pl/aplikacje-natywne-internetowe-hybrydowe/ [07.01.2019]

[11] Harbuz M.: Czym jest aplikacja hybrydowa? www.x-coding.pl/blog/developers/czym-jest-aplikacja-hybrydowa/ [08.01.2019]

[12] comScore, Inc: The 2016 U.S. Mobile App Report. comScore, Inc. 2016.

[13] Alter T.: Building Progressive Web Apps. O'Reily Media, Inc., 2017.

[14] Suchodolski P.: PWA (Progressive Web App) – pierwsze kroki. www.sunscrapers.com/blog/pwa-pierwsze-kroki/ [07.01.2019]

[15] Archibald J.: Introducing Background Sync. www.developers.google.com/web/updates/2015/12/background-sync [07.01.2019]

[16] Gauntt M.: Adding Push Notifications to a Web App. www.developers.google.com/web/fundamentals/codelabs/push-notifications/ [07.01.2019]

[17] Progressive Web App Checklist. www.developers.google.com/web/progressive-web-apps/checklist [08.01.2019]