Programowanie aplikacji mobilnych (1)

Instalacja narzędzi programistycznych

Większość urządzeń elektronicznych, których używamy, wymaga interakcji z użytkownikiem, a więc interfejsu sterującego. Typowym sposobem, po jaki sięgają inżynierowie elektronicy, jest montaż przycisków i diod lub wyświetlaczy, które pozwalają na przekazywanie komendy do urządzenia oraz odczytywanie jego stanu. A gdyby tak zrezygnować z tych elementów i sterować zaprojektowanym urządzeniem za pomocą smartfona? Rozpoczynamy kurs, który – mamy nadzieję – umożliwi czytelnikom tworzenie interfejsów użytkownika, a w praktyce nawet samodzielnych aplikacji, które będzie można uruchamiać na większości popularnych mobilnych systemów operacyjnych: Androidzie, iOS, Windows Phone, BlackBerry OS, a nawet na Firefox OS czy Tizen.

Ten kurs został napisany dla elektroników i przez elektronika, co istotnie różni go od wszelkich kursów programowania aplikacji mobilnych dostępnych w Internecie. Wybrano taki zestaw narzędzi, aby tworzenie programów było jak najłatwiejsze, kod przenośny oraz żeby wyjaśnić wszelkie wątpliwości, jakie mogą pojawić się w głowie inżyniera niezaznajomionego z wykorzystywanymi technikami programowania.

Wybrane narzędzia mają pewne ograniczenia, ale moim zdaniem idealnie nadają się do tworzenia interfejsów użytkownika, czyli aplikacji, które same w sobie nie wymagają ogromnej mocy obliczeniowej, ani nie realizują bardzo skomplikowanych zadań. Mają natomiast pozwolić na szybkie i łatwe utworzenie programu sterującego, którego oprawę graficzną można w razie potrzeby upiększyć, tak by nie ustępowała najnowszym trendom w tworzeniu graficznych interfejsów użytkownika. Zakładam, że czytelnicy, po dotarciu do końca kursu, będą umieli stworzyć własne programy, korzystające z podzespołów smartfona czy tabletu, takich jak np.:

- akcelerometr,
- GPS,
- kamera,
- mikrofon,
- głośniki,
- żyroskop,
- interfejs sieciowy (Wi-Fi lub komórkowy),
- interfejs Bluetooth,
- interfejs USB.

Pozwoli to na znaczące zwiększenie atrakcyjności tworzonych projektów urządzeń elektronicznych, usprawnienie ich obsługi, a nawet zmniejszenie kosztów ich produkcji. W końcu rezygnacja z wyświetlacza i klawiatury, które zastąpi ekran dowolnego telefonu komórkowego lub tabletu, pozwala na osiągnięcie dużych oszczędności.

Co ważne, praktycznie wszystkie narzędzia używane w trakcie kursu są bezpłatne. Jedyny problem może stanowić przygotowanie aplikacji na system iOS, która – aby była dostępna dla każdego urządzenia z tym systemem – musi zostać wprowadzona do sklepu Apple App Store, co może wiązać się z dodatkowymi kosztami. Natomiast ze względu na popularność i elastyczność poszczególnych platform, większość kursu będzie koncentrowała się na systemie Android, a dopiero później

Tabela 1. Możliwość kompilacji aplikacji korzystających z Apache Cordova dla poszczególnych systemów mobilnych, w zależności od systemu operacyjnego środowiska deweloperskiego. Platforma Browser umożliwia symulację mobilnego systemu operacyjnego w przeglądarce internetowej Google Chrome.

Platforma docelowa/deweloperska	Windows	Linux	Mac OS
Amazon FireOS	tak	tak	tak
Android	tak	tak	tak
BlackBerry 10	tak	nie	tak
Firefox OS	tak	tak	tak
iOS	nie	nie	tak
Ubuntu Touch	nie	tak	nie
Windows Phone 8	tak	nie	nie
Windows (8.0, 8.1, Windows Phone 8.1)	tak	nie	nie
Tizen	tak	tak	nie
Browser	tak	tak	tak

pokażę, jak przenieść kod do aplikacji pozostałych systemów operacyjnych.

Co musisz już umieć?

Jednym ze sposobów na stworzenie uniwersalnej aplikacji, działającej na wielu różnych mobilnych systemach operacyjnych, jest wykonanie jej w tzw. technologii webowej. Jednakże przygotowanie strony internetowej, która byłaby interfejsem do jakiegoś urządzenia sieciowego, uruchamianym w przeglądarce internetowej to coś znacząco innego niż stworzenie samodzielnej aplikacji, instalowanej na telefonie i korzystającej z jego podzespołów. W niniejszym kursie pokazujemy to drugie rozwiązanie, które pozwala na stworzenie w pełni profesjonalnej aplikacji. Niemniej będziemy korzystali z języków programowania, stosowanych w stronach internetowych: z HTMLu i JavaScriptu. Oba są jednymi z prostszych do nauki, a ponadto w przypadku HTMLu wystarczy tylko jego podstawowa znajomość. Wyższy stopień znajomości języka HTML oraz znajomość CSS będą pomocne przy rozbudowie szaty graficznej aplikacji. Natomiast znajomość języka JavaScript będzie konieczna, choć postaramy się wyjaśnić specyfikę i sposoby używania bibliotek JavaScriptowych, z jakich będziemy korzystać.

Czym jest Cordova?

W jaki sposób umiejętność tworzenia stron internetowych w HTMLu i JavaScripcie ma umożliwić stworzenie uniwersalnej aplikacji na smartfony, gdzie programy uruchamiane na Androidzie są napisane w Javie, a iOS bazuje na opracowanym przez Apple języku Objective-C? Kluczem do sukcesu jest platforma Cordova, której twórcami i opiekunami są członkowie Apache Software Foundation, organizacji, która rozwija m.in. najbardziej popularny, otwarty serwer HTTP.

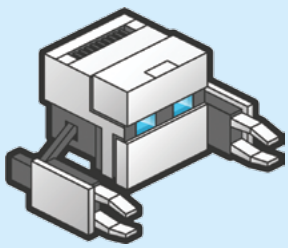
Apache Cordova to zestaw bibliotek i interfejsów programistycznych, które umożliwiają dostęp do zasobów urządzeń mobilnych, pracujących pod kontrolą różnych systemów operacyjnych. Biblioteki zostały przygotowane w różnych wersjach, dla poszczególnych systemów operacyjnych, ale zachowano w nich identyczny sposób wywoływania poleceń i interfejs programistyczny, dzięki czemu niezależnie od systemu, na którym będzie uruchamiana aplikacja, wszystkie funkcje wywoływane są tak samo. Istnieją tylko pojedyncze wyjątki, związane z niedostępnością nielicznych funkcji w niektórych systemach operacyjnych oraz różnice w definiowaniu ustawień i konfiguracji aplikacji.

Aplikacja napisana w HTMLu i JavaScripcie, korzystająca z bibliotek Apache Cordova oraz dowolnych innych bibliotek JavaScriptowych, jest następnie kompilowana pod kątem wybranego systemu operacyjnego i w ten sposób powstaje plik, który można np. wgrać do telefonu z Androidem, czy nawet umieścić w sklepie App Store albo Google Play. Naturalnie kod napisany w JavaScripcie, który jest językiem interpretowanym na żywo, będzie pracować znacznie mniej wydajnie, niż program stworzony bezpośrednio w potężnym Objective-C, ale w pełni wystarczy do realizacji nawet bardzo dynamicznych interfejsów użytkownika.

Środowiska deweloperskie a docelowe

Napisaliśmy, że kurs będzie opierał się o wykorzystanie darmowych narzędzi, ale już na wstępie zrobimy pewien wyjątek, gdyż skorzystamy z systemu operacyjnego Windows 7, który prawdopodobnie będzie bliższy wielu inżynierom elektronikom, niż Linux. Co więcej, zależnie od systemu, na którym jest kompilowany projekt oparty o platformę Cordova, są różne możliwości wyboru systemów, na których aplikacja będzie mogła być uruchamiana. Inaczej mówiąc – niezależnie czy posługujemy się systemem Windows, Linux, czy Mac OS, możemy rozwijać program na dowolny z mobilnych systemów operacyjnych, ale samo zbudowanie pliku wykonywalnego (instalowalnego) jest już ograniczone. Korzystając z Windowsa można kompilować aplikacje dla dowolnych systemów operacyjnych, za wyjątkiem iOS. Korzystając z Mac OS można kompilować aplikacje dla dowolnych systemów operacyjnych, za wyjątkiem Windows Phone i Windows. A korzystając z Linuxa nie można skompilować programów ani pod systemy z rodziny Windows, ani pod Mac OS. Ograniczenia te zostały przedstawione w **tabeli 1**. Jest to jeden z czynników, dla których kurs koncentruje się na aplikacjach kompilowanych na Androida, który nie dość, że jest bardzo popularny, to nie wprowadza takich ograniczeń, jak iOS, czy Windows Phone.

PhoneGap i PhoneGap Build



Taka sytuacja może sugerować, że posiadacz komputera z systemem Linux nie będzie w stanie przygotować programu na potrzeby bardzo popularnych przecież iPadów, iPhone'ów czy iPodów, ale i na to jest rozwiązanie. Nie trzeba kupować MacBooka, tylko wystarczy skorzystać z usługi PhoneGap Build, świadczonej przez firmę Adobe. Usługa ta oferowana jest w wersjach: bezpłatnej i płatnej, które różnią się ograniczeniami. Przygotowane pliki projektu są ładowane na serwery Adobe i kompilowane w chmurze, w efekcie czego użytkownik otrzymuje pliki gotowe do wgrania do smartfona, czy do umieszczenia

Tabela 2. Ograniczenia usługi Adobe PhoneGap Build, w zależności od rodzaju wybranej subskrypcji; stan na styczeń 2015 r.

Cecha/Rodzaj subskrypcji	Bezpłatna	Płatna subskrypcja	Subskrybenci Adobe Creative Cloud
Kompilowanie aplikacji open-source, których kod będzie dostępny publicznie na serwerach Github	nielimitowane		
Kompilowanie aplikacji prywatnych, których kod nie będzie dostępny publicznie	1 aplikacja	25 aplikacji	
Maksymalny rozmiar aplikacji	50 MB	100 MB	1 GB
Możliwość korzystania z oficjalnych bibliotek Cordova, opracowanych przez Apache	tak		
Możliwość korzystania z bibliotek Cordova, opracowanych przez inne firmy niż Apache	tak		
Możliwość wgrywania dowolnych bibliotek (w tym własnych)	nie	tak	
Możliwość pracy grupowej poprzez chmurę Adobe	nielimitowana		
Koszt usługi PhoneGap Build	bezpłatny	9,99 USD/miesiąc	konieczność utrzymania subskrypcji Adobe Creative Cloud

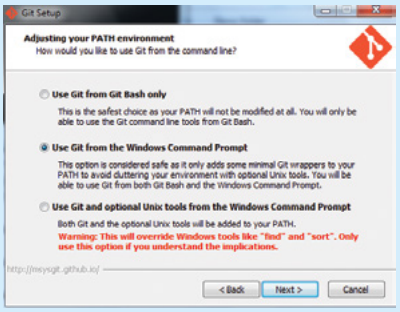
w sklepie z aplikacjami. Różnice pomiędzy wersjami bezpłatną a płatną polegają na ograniczeniu dopuszczalnej wielkości aplikacji (50 MB i 100 – 1024 MB, odpowiednio), oraz w tym że w wersji darmowej aplikacja nie może korzystać z bibliotek stworzonych przez użytkownika, innych niż oferowane przez Adobe. Co więcej, w ramach jednego konta Adobe, poprzez PhoneGap Build można skompilować tylko jedną aplikację prywatną, czyli taką, która nie stanie się publicznie dostępna wraz z jej kodem źródłowym. Różnice w opcjach wersji płatnej i bezpłatnej usługi PhoneGap Build pokazano w **tabeli 2**.

Warto też zauważyć, że PhoneGap Build opiera się o darmową platformę PhoneGap, która jest pewnego rodzaju dystrybucją platformy Cordova. W praktyce PhoneGap jest prawie identyczna z Cordovą i teoretycznie może zawierać dodatkowe elementy, które firma Adobe dodała do pakietu instalacyjnego. Drobne różnice pojawiają się też na etapie instalacji lub ładowania bibliotek, ale w praktyce najczęściej sprowadzają się do zastąpienia słowa „cordova" wyrazem „phonegap", a parametry używanych poleceń się nie zmieniają. Przygotowując sobie środowisko programistyczne należy pobrać i zainstalować tylko jedną z tych platform. W naszym przypadku będziemy korzystać z Cordovy, ale postaramy się wskazać, w których momentach występują różnice pomiędzy wybraną platformą a PhoneGapem.

Podstawowa instalacja

Zaczynamy od instalacji uniwersalnej, minimalnej wersji platformy Cordova, którą następnie będzie można rozbudować o dodatkowe narzędzia, ułatwiające programowanie. Uwaga – całość, choć bez narzędzi do kompilacji pod inne systemy operacyjne niż Android, wymaga około 21 GB przestrzeni dyskowej!

Aby zainstalować Cordovę, należy wcześniej posiadać dwa dodatkowe narzędzia: środowisko NodeJS



Rysunek 1. Instalowanie programu GIT

i program do korzystania z zasobów Git, czyli środowiska do kontroli wersji, które często używane jest przez dostawców bibliotek, przydatnych podczas programowania.

NodeJS pobieramy z witryny ***nodejs.org*** i instalujemy, jak każdy normalny program. W naszym wypadku pobraliśmy NodeJS w wersji 0.10.35 na systemy 64-bitowe i skorzystaliśmy z domyślnych opcji instalacji.

Następnie pobieramy pakiet Git ze strony ***git-scm.com*** (w naszym przypadku w wersji 1.9.5-preview20141217) i również instalujemy go na komputerze, przy czym dla ułatwienia na przyszłość, w oknie wyboru informacji wpisywanych do systemowej zmiennej środowiskowej PATH, wskazaliśmy opcję „Use Git from the Windows Command Prompt" (**rysunek 1**), która pozwoli nam później bezproblemowo, samodzielnie pobierać dodatkowe biblioteki z dowolnych repozytoriów GIT.

Mając zainstalowany NodeJS możemy przystąpić do instalacji platformy Cordova, która została przygotowana w postaci pakietu środowiska NodeJS. Do instalacji pakietów NodeJS służy komenda npm, wywołująca menedżera pakietów. Uruchamiamy więc linię poleceń i z dowolnej ścieżki (komenda npm została w trakcie instalacji NodeJS dodana do systemowych zmiennych środowiskowych) wydajemy polecenie:

npm install –g cordova

Cordova zostaje wtedy automatycznie pobrana z Internetu i zainstalowana jako moduł w odpowiednim katalogu środowiska NodeJS (w przypadku podanego parametru „–g", w taki sposób by Cordova była dostępna z dowolnej ścieżki). Wszelkie biblioteki zależne zostaną pobrane automatycznie i również zainstalowane. Wydane polecenie instaluje najnowszą dostępną wersję Cordovy, w efekcie czego u nas pobrała się wersja 4.1.2. Informacja na ten temat jest podawana przez menedżera

pakietów npm, ale można ją też sprawdzić już po instalacji, wpisując polecenie:

cordova –v

Istnieje też możliwość wymuszenia instalacji konkretnej wersji poprzez wydanie polecenia np.:

npm install –g cordova@4.1.2

W ten sposób czytelnicy mogą dokładnie podążać za kursem, korzystając z identycznej wersji oprogramowania. W praktyce niewiele nowsza wersja nie powinna sprawiać znaczącej różnicy, ale nasze doświadczenia pokazują, że tak złożone środowisko deweloperskie, jakie docelowo przygotujemy w ramach kursu, wrażliwe jest na drobne zmiany w wersjach bibliotek i narzędzi, co dla niedoświadczonego programisty może stanowić niemały problem.

Pierwsza aplikacja

Minimalne środowisko deweloperskie jest już gotowe. Jak na razie można z niego korzystać jedynie z linii poleceń, która stanowi całkiem wygodny sposób rozpoczynania budowy aplikacji. Zaczynamy od stworzenia katalogu, w którym będziemy przechowywać nasze projekty. U nas będzie to ścieżka:

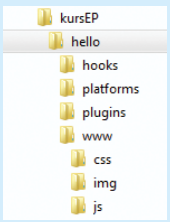
c:\kursEP

przechodzimy do niej i z linii poleceń wywołujemy komendę:

cordova create hello com.example.hello HelloWorld

Nakazuje ona pobranie odpowiednich bibliotek i wygenerowanie podkatalogu hello, w którym znajduje się projekt aplikacji o nazwie „HelloWorld”. Podanie dwóch ostatnich parametrów (w tym przypadku com.example.hello i HelloWorld) jest opcjonalne, ale warto już na tym etapie wpisać oba parametry. Pierwszy z nich to identyfikator aplikacji, zapisany w odwrotnym formacie domenowym. Drugi to właściwa nazwa aplikacji, która będzie wyświetlać się w wielu miejscach, jako tytuł całego, powstającego programu. Jeśli odpowiednie nazwy podamy już teraz, nie będziemy musieli ich następnie ręcznie zmieniać w wielu miejscach w kodzie źródłowym.

Nasza pierwsza aplikacja już działa, choć jeszcze praktycznie nic nie robi i można ją uruchomić tylko i wyłącznie za pomocą przeglądarki internetowej. Jej strona główna, czyli plik zawierający kod uruchamiający aplikację w przeglądarce, znajduje się w podkatalogu **www** aplikacji i nosi nazwę **index.html**. Zresztą cały podkatalog **www** zawiera właściwy kod aplikacji, podczas gdy pozostałe podkatalogi będą zawierać różne biblioteki. Natomiast w pliku **config.xml**, umieszczonym bezpośrednio w katalogu głównym aplikacji, znajdziemy parametry tworzonego programu, takie jak jej nazwa, opis, dane autora oraz wskazanie na plik startowy.



Rysunek 2. Struktura katalogowa nowego projektu Cordovy

Dla osób niezorientowanych w tworzeniu stron internetowych wyjaśniamy, że podkatalogi **css**, **img** i **js**, umieszczone w katalogu **www** służą do przechowywania różnego rodzaju zasobów i kodu źródłowego, zgodnie z popularną konwencją programowania. W podkatalogu **css** znajdą się pliki styli, służące do wygodnego określania wyglądu elementów wyświetlanych w aplikacji, katalog **img** służy do przechowywania plików graficznych, a **js** do kodu JavaScript. Uzyskana struktura katalogowa została przedstawiona na **rysunku 2**.

Dodanie obsługi mobilnych systemów operacyjnych

Po przejściu do katalogu głównego stworzonej aplikacji sprawdźmy, dla jakich mobilnych systemów operacyjnych możemy kompilować aplikacje na danym komputerze. W tym celu wydajemy polecenie:

cordova platform ls

Jeżeli wszystko poszło zgodnie z planem, powinniśmy nie mieć jeszcze żadnej platformy zainstalowanej dla naszej aplikacji, ale na liście dostępnych platform dla komputera z systemem Windows 7 powinny pojawić się:

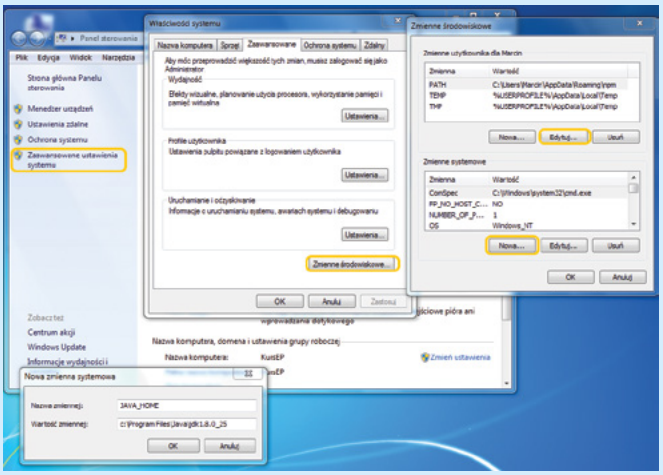
- amazon-fireos,
- android,
- blackberry10,
- browser,
- firefoxos,
- windows,
- windows8,
- wp8.

Android, a więc Java

My zaczniemy od Androida, ale nim to zrobimy, musimy zainstalować środowisko deweloperskie Java, czyli JDK. Pobieramy je ręcznie z witryny firmy Oracle (*www.oracle.com*), z działu Downloads->Java SE, gdzie wybieramy wersję Java Platform (JDK) 8 – w naszym przypadku Update 25 dla komputerów Windows z systemem 64-bitowym (plik jdk-8u25-windows-x64.exe). Instalacja przebiega z domyślnymi ustawieniami i obejmuję instalację także środowiska uruchomieniowego Javy (JRE), gdyż nie było ono wcześniej zainstalowane na naszym komputerze. Następnie należy dodać do listy zmiennych systemowych zmienną **JAVA_HOME**, której wartość ma wskazywać ścieżkę katalogu, w którym zainstalowany jest Java Development Kit. W naszym wypadku jest to:

c:\Program Files\Java\jdk1.8.0_25

Wartość tę wpisujemy klikając prawym przyciskiem myszy na „Mój Komputer” i wybierając „Właściwości”. Następnie z menu po lewej stronie wybieramy „Zaawansowane ustawienia systemu” i w zakładce „Zaawansowane” klikamy przycisk „Zmienne środowiskowe”. W dziale „Zmienne systemowe”



Rysunek 3. Dodawanie i edycja systemowych zmiennych środowiskowych

klikamy „Nowa…” i jako nazwę zmiennej wpisujemy **JAVA_HOME**, a jako wartość, odpowiednią ścieżkę, tak jak zostało to pokazane na **rysunku 3**.

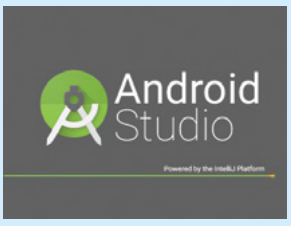
ANT

Teraz kolej na konieczne narzędzie, usprawniające kompilowanie. Apache ANT zastępuje program Make i jest oparte o Javę, a konfigurację kompilacji odczytuje z plików XML. Było ono częścią niektórych wersji Cordovy i PhoneGap, ale w pobranej przez nas, najnowszej dystrybucji, Apache ANT się nie znalazło, dlatego musimy je zainstalować samodzielnie.

Program pobieramy ze strony ***ant.apache.org***, przechodząc do działu „Download->Binary Distributions”. Szukamy linku do „main distribution directory” lub od razu przechodzimy do adresu ***www.apache.org/dist/ant/binaries/***. Następnie wybieramy jedyny plik zip w zestawieniu, który zawiera wersję ANT dla Windows. W naszym przypadku jest to **apache-ant-1.9.4-bin.zip**. Pobrany plik rozpakowujemy do wybranego katalogu. My wrzuciliśmy go do **C:\Program Files (x86)**. Program nie wymaga instalacji, ale konieczne jest dopisanie ścieżek do zmiennych środowiskowych. Należy ustalić zmienną **ANT_HOME**, kierującą do głównego katalogu programu ANT oraz do zmiennej **PATH** dodać ścieżkę do podkatalogu **bin**. W naszym przypadku wygląda to tak: **ANT_HOME** to **c:\Program Files (x86)\apache-ant-1.9.4**, a zmienna **PATH** została wydłużona o ciąg: **%ANT_HOME%\bin**, oddzielony średnikiem.

Android SDK i ADT

To jeszcze nie wszystko. Konieczne są też narzędzia deweloperskie dla Androida – Android SDK. Można je pobrać samodzielnie lub w ramach IDE (zintegrowanego środowiska deweloperskiego – w praktyce edytora, kompilatora itd.). Tu możliwości jest kilka. Osoby korzystające z popularnej platformy Eclipse być może preferowałyby skorzystanie z pluginu ADT (Android Development Tools) do Eclipse’a, ale od pewnego czasu oficjalnym narzędziem dla Androida jest Android Studio. Ponieważ w dalszej części kursu edytor będzie przydatny, użyjemy teraz Android Studio, a obsługę ADT i Eclipse postaramy się pokazać w jednej z kolejnych części.



Android Studio pobieramy z adresu ***developer.android.com/sdk***. W naszym przypadku jest to plik **android-studio-bundle-135.1641136.exe**. Może to chwilę potrwać, gdyż zajmuje on ponad 800 MB. Instalujemy z domyślnymi opcjami – w tym z emulatorem Androida. Podczas pierwszego uruchomienia pomijamy import danych z poprzedniego środowiska Android Studio oraz czekamy na aktualizację narzędzi SDK. W naszym przypadku pobrała się wersja Android SDK Tools 24.0.2. oraz Android Support Repository revision 11. Po pełnym ukończeniu instalacji zamykamy Android Studio, gdyż teoretycznie nie będzie nam potrzebne do pierwszej kompilacji przykładowej aplikacji. Natomiast konieczne będzie dopisanie do zmiennych środowiskowych kolejnej ścieżki. Tym razem dodajemy zmienną **ANDROID_HOME**, kierującą do katalogu głównego Android Studio – w naszym przypadku jest to:

C:\Users\<username>\AppData\Local\Android\sdk

gdzie w miejsce *<username>* wpisujemy nazwę użytkownika Windows.

Dodajemy też dwie ścieżki systemowe do zmiennej **PATH**, tak aby kierowały do katalogów z plikiem **android.bat**. W tym celu oknie „Zmienne środowiskowe” edytujemy zmienną **PATH** i dopisujemy na jej końcu, po średniku, wartość:

C:\Users\<username>\AppData\Local\Android\sdk\tools;C:\Users\<username>\AppData\Local\Android\sdk\platform-tools

lub inną, odpowiadającą lokalizacji tych katalogów. Oczywiście w miejsce <username> wpisujemy nazwę użytkownika Windows. **Uwaga!** Te katalogi będą się znacząco różnić, w zależności od wersji SDK Androida i w razie trudności z ich zlokalizowaniem warto przeszukać dysk w poszukiwaniu pliku **android.bat**.

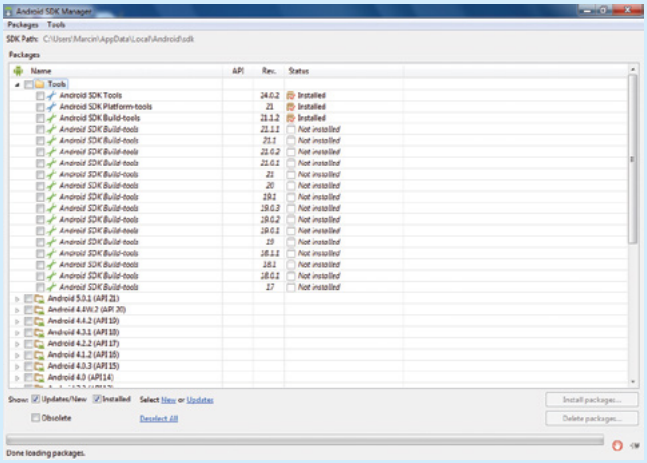
Po wpisaniu zmiennych środowiskowych, dla pewności warto zrestartować komputer.

Tabela 3. Wersje Androida, ich nazwy kodowe i odpowiadające im poziomy API (API Level). Poziom 20 różni się od poziomu 19 dodatkową obsługą urządzeń noszonych, takich jak np. inteligentne zegarki

API Level	Android	Nazwa
1	1.0	
2	1.1	
3	1.5	Cupcake
4	1.6	Donut
5	2.0	Eclair
6	2.0.1	
7	2.1	
8	2.2-2.2.3	Froyo
9	2.3-2.3.2	Gingerbread
10	2.3.3-2.3.7	
11	3.0	Honeycomb
12	3.1	
13	3.2	
14	4.0-4.0.2	Ice Cream Sandwich
15	4.0.3-4.0.4	
16	4.1	Jelly Bean
17	4.2	
18	4.3	
19	4.4	KitKat
20	4.4 (+wearable)	
21	5.0	Lollipop

Konflikty wersji

Teoretycznie mamy już zainstalowane wszystko co potrzebne, ale w praktyce najprawdopodobniej wystąpi konflikt wersji oprogramowania. Apache Cordova i Android rozwijane są przez niezależne zespoły i niezależnie od siebie wydawane. Dlatego najczęściej Cordova nie jest przystosowana do obsługi najnowszej wersji Androida, która automatycznie pobiera się wraz z Android Studio. Tymczasem od strony programistycznej różnice pomiędzy poszczególnymi wersjami Androida układają się nieco inaczej, niż wynikałoby to z powszechnie



Rysunek 4. Menedżer SDK Androida

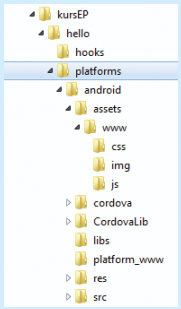
stosowanego nazewnictwa. Dla programisty ważny jest poziom API (API Level), wspierany przez dane urządzenie z Androidem. Poziomy i odpowiadające im wersje Androidów zostały zebrane w **tabeli 3**.

W naszym przypadku pobrane SDK jest dla 21 poziomu API, czyli Androida 5.0 Lollipop. Natomiast Cordova 3.6.4 została przystosowana do SDK Androida 4.4 Kitkat, a więc 19 poziomu API. Problem ten można rozwiązać na dwa sposoby:

- zmieniając poziom API w ustawieniach Cordovy,
- instalując starszą wersję SDK Androida.

Pierwsza opcja jest szybsza i teoretycznie powinna działać. Ogólną zasadą jest bowiem, że oprogramowanie przystosowane do pracy na starszej wersji Androida będzie też działać na wersji nowszej, a więc Cordova dla Androida 4.4 powinna poprawnie pracować na Androidzie 5.0. Aby dokonać takiej zmiany ustawień, należy znaleźć plik **project.properties**, w którym zapisana jest wersja Androida, preferowana przez Cordovę. W naszym przypadku plik znalazł się w katalogu:
c:\Users\Marcin\.cordova\lib\npm_cache\cordova-android\3.6.4\package\framework

Wystarczy podmienić wartość przypisaną do stałej **target** na **android-21**. My jednak na wszelki wypadek wybierzemy opcję b), która przy okazji pozwoli pokazać, jak dodawać i zmieniać platformy, dla których kompilowane i testowane są aplikacje.



Rysunek 5. Struktura katalogowa projektu Cordovy po dodaniu do niej obsługi platformy androidowej

Potrzebny będzie Android SDK Manager, którego można uruchomić albo z poziomu Android Studio (z głównego ekranu wybieramy „Configure->SDK Manager"), albo samodzielnie z katalogu głównego SDK Androida:
c:\Users\Marcin\AppData\Local\Android\sdk\SDK Manager.exe
W praktyce często działa też polecenie:
android

Po uruchomieniu menedżera automatycznie pojawi się lista zainstalowanych wersji SDK (**rysunek 4**). Domyślnie też menedżer sugeruje instalację dodatkowych elementów, które w tej chwili nie będą nam potrzebne, więc je odznaczamy. Zamiast tego zaznaczamy pozycje: Android SDK Build-tools w wersji 19 oraz wszystkie pozycje z działu Android 4.4.2 (API19), po czym klikamy „Install" (9 pakietów), zgadzamy się na licencje i potwierdzamy instalację. Ściąganie pakietów może trochę potrwać, a po zakończeniu instalacji zamykamy menedżera.

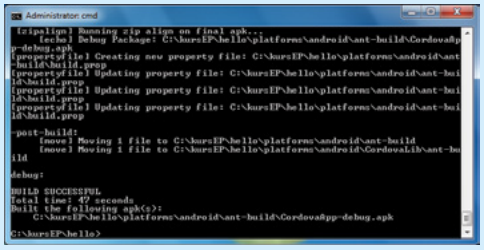
Kompilowanie aplikacji dla Androida

Ponownie uruchamiamy linię poleceń i przechodzimy do katalogu głównego utworzonej, przykładowej aplikacji. Możemy już dodać do niej obsługę platformy Android na poziomie API 19. W tym celu wpisujemy:
cordova platform add android

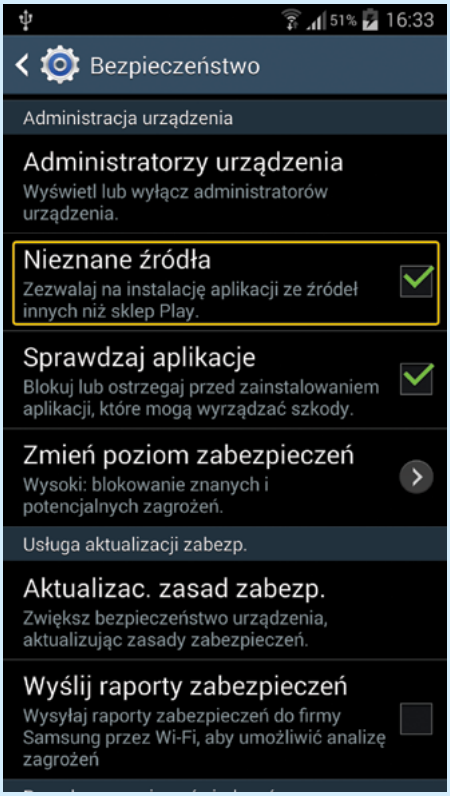
Spowoduje to pojawienie się w katalogu **platforms** podkatalogu **android**, zawierającego kompletny zestaw plików potrzebnych do skompilowania aplikacji pod Androida 4.4 (**rysunek 5**). Warto zwrócić uwagę na uzyskaną strukturę katalogową. Katalogi **cordova** i **CordovaLib** zawierają biblioteki Cordovy. Pierwszy z nich obejmuje biblioteki napisane w JavaScripcie, związane z środowiskiem NodeJS. Drugi – biblioteki Javy, a więc natywne dla Androida, które stanowią trzon aplikacji androidowej. Katalog **res** zawiera zasoby aplikacji androidowej – w tym przypadku głównie ustawienia i grafiki (ikonki). Natomiast katalog **assets** zawiera podkatalog **www**, do którego przekopiowane zostały kluczowe pliki kodu źródłowego, z katalogu **www**, znajdującego się w katalogu głównym aplikacji. W przypadku korzystania z edytorów takich jak Eclipse, podkatalog **www** w katalogu **assets** będzie domyślnie niewidoczny, gdyż teoretycznie nie należy zmieniać znajdującej się tam treści. Ma ona być każdorazowo na nowo kopiowana z głównego katalogu **www**, tak by nie różniła się niczym, niezależnie od platformy, na którą aplikacja jest kompilowana. Zaleca się, by wszelkie zmiany w kodzie wykonywać na plikach w głównym katalogu **www**. Wszelkie zmiany w plikach w podkatalogach katalogu **platforms** należy robić pod kątem konkretnych mobilnych systemów operacyjnych (uważając, by nie zostały automatycznie nadpisane), co bywa konieczne, jeśli stosuje się zintegrowane środowiska deweloperskie, jakie jak np. Eclipse czy Xcode. Natomiast my skompilujemy nasz pierwszy program, korzystając z mechanizmu Cordova CLI (Cordova Command Line Interface), bez potrzeby uciekania się do zewnętrznych edytorów.

Aby przygotować kod do kompilacji wywołujemy polecenie:
cordova build,
które automatycznie buduje aplikacje dla wszystkich zainstalowanych platform. Czas pierwszej kompilacji dla Androida może wynieść około 30…60 sekund (**rysunek 6**), a rezultatem jest przede wszystkim plik o rozszerzeniu APK.

Warto jeszcze zwrócić uwagę na pliki wygenerowane przez polecenie cordova build. Znalazły się one m.in. w dwóch nowych podkatalogach katalogu **platforms\android**. Kluczowy jest **ant-build**, w którym znajduje

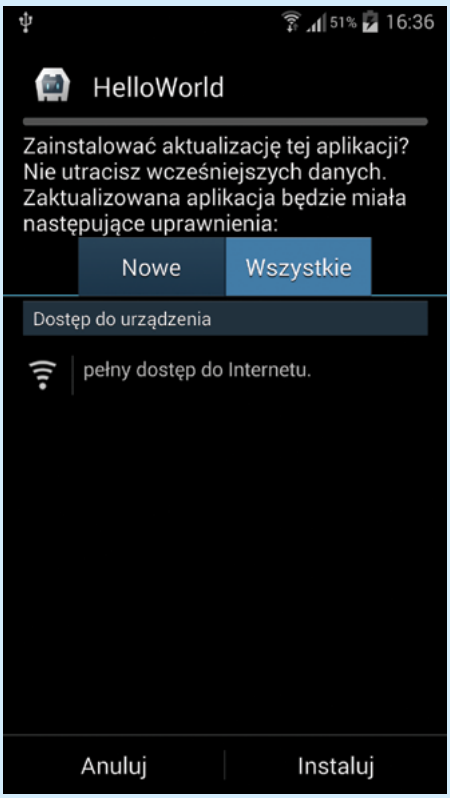


Rysunek 6. Pierwsza kompilacja projektu Cordova na Androida

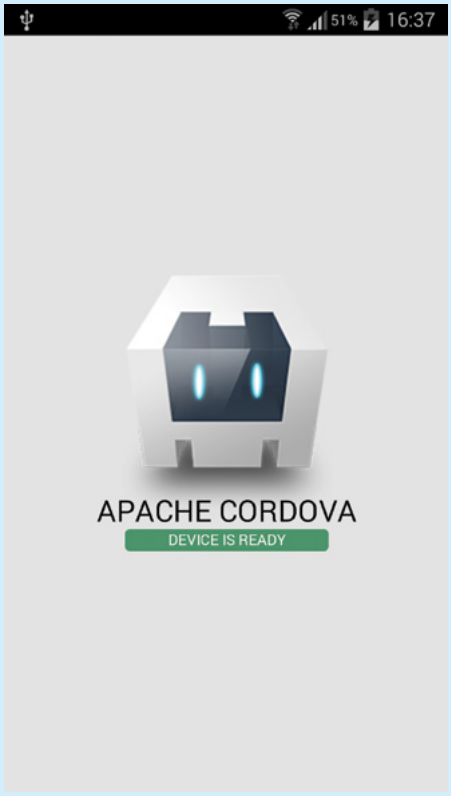


Rysunek 7. Instalowanie własnej aplikacji wymaga ustawienia w telefonie pozwolenia na aplikacje z nieznanych źródeł

się gotowa aplikacja. W naszym przypadku jest to plik **CordovaApp-debug.apk**. Plik ten można teraz wgrać na smartfon lub tablet i zainstalować. Ważne, by w opcjach urządzenia zezwolić na instalację aplikacji z nieznanych źródeł (**rysunek 7**), czyli takich, które nie



Rysunek 8. Instalowanie utworzonej aplikacji na telefonie



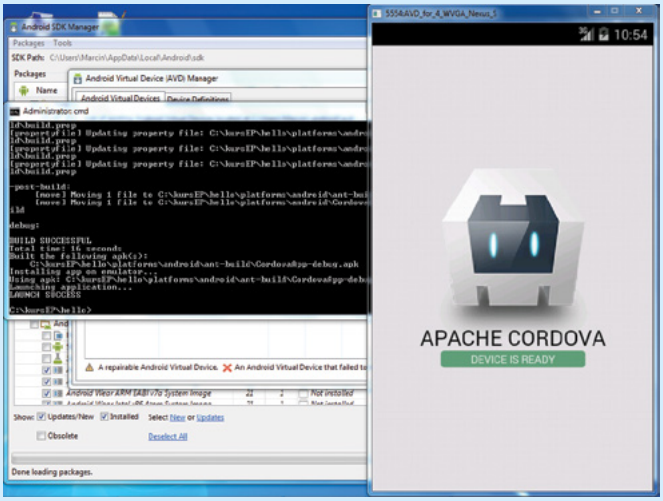
Rysunek 9. Nasza aplikacja pomyślnie uruchomiona na telefonie

pochodzą np. ze sklepu Google Play. W przeciwnym wypadku aplikacji nie da się wykonać tego kroku. Po zainstalowaniu naszej aplikacji (**rysunek 8**) można ją uruchomić (**rysunek 9**).

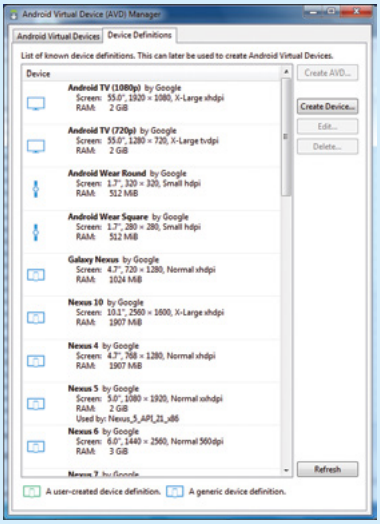
Emulowanie Androida

Skompilowanie i uruchomienie aplikacji na telefonie to jeszcze nie koniec przygotowywania środowiska do dalszego programowania. Warto zadbać o odpowiednie emulatory, które pozwolą na testowanie programów na różnego rodzaju telefonach, tabletach, a nawet inteligentnych telewizorach czy zegarkach. Android SDK jest dostarczany wraz z całkiem dobrym, bezpłatnym emulatorem. Domyślny emulator androidowy z SDK można uruchomić z linii poleceń Cordovy, korzystając z komendy:

cordova emulate android

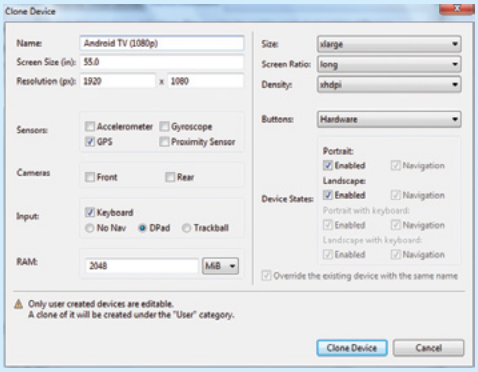


Rysunek 10. Domyślny symulator nowoczesnego telefonu z Androidem

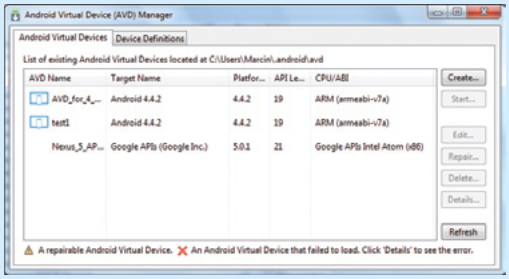


Rysunek 11. Menedżer definicji sprzętu wirtualnych domyślnego emulatora Androida

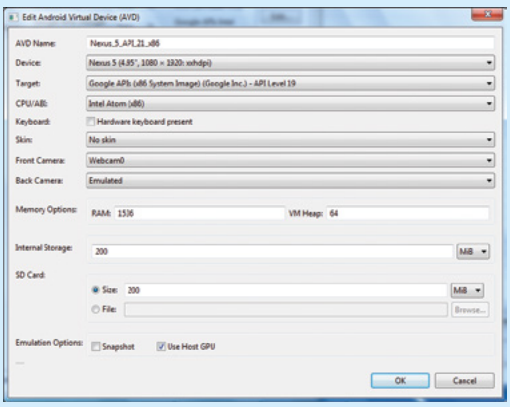
Trzeba jednak trochę poczekać na włączenie się emulatora (**rysunek 10**), a ponadto będzie to najnowsza wersja dostępnego systemu Android, czyli w naszym przypadku Nexus 5 z API 21 i procesorem x86. Może to powodować problemy na niektórych starszych komputerach, albo np. w sytuacji, gdy całe środowisko programistyczne i system operacyjny mamy zainstalowane na maszynie wirtualnej. Warto więc nauczyć się konfigurowania innych modeli urządzeń w emulatorze.



Rysunek 12. Definiowanie sprzętu emulowanego urządzenia androidowego



Rysunek 13. Lista profili sprzętowo-programowych emulowanych urządzeń androidowych

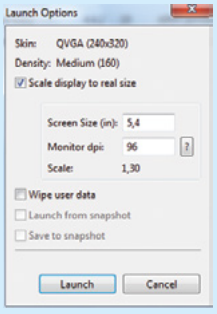


Rysunek 14. Definiowanie profilu sprzętowo-programowego emulowanego urządzenia androidowego

W tym celu wywołajmy znany już Android SDK Manager, np. korzystając z komendy:

android

wpisanej w linii poleceń. Następnie z menu „Tools" wybieramy „Manage AVDs", czyli okno zarządzania urządzeniami wirtualnymi. Znajdziemy tam dwie zakładki. „Device Definitions" (**rysunek 11**) zawiera definicje sprzętu, określające urządzenia o konkretnych: rozdzielczości, przekątnej i kształtcie ekranu, gęstości upakowania pikseli, rodzaju wbudowanych przycisków (programowe, sprzętowe), wbudowanych komponentach, takich jak sensory (akcelerometr, żyroskop, GPS, czujnik zbliżeniowy), kamerach, klawiaturach, dostępnej pamięci RAM i obsługiwanych trybach pracy (**rysunek 12**).



Rysunek 15. Opcje uruchamiania emulowanego urządzenia androidowego

Na podstawie tych definicji sprzętu można utworzyć konkretne urządzenia, zebrane w zakładce „Android Virtual Devices" (**rysunek 13**). Wymaga to podania poziomu obsługiwanego API, rodzaju architektury procesora (ARM lub x86), rozdzielczości interfejsu graficznego, źródła sygnału kamery w emulowanym urządzeniu, sposobu podziału pamięci RAM, wielkości dostępnej, wbudowanej pamięci Flash oraz pojemności włożonej karty SD (**rysunek 14**).

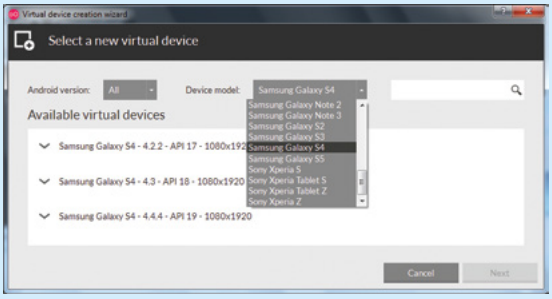
Gotowe urządzenia wirtualne można też uruchomić samodzielnie, bez włączania na nich żadnej konkretnej aplikacji. Jednocześnie emulator pozwala na przeskalowanie wyświetlanego obrazu do rozmiarów rzeczywistych, w oparciu o rozdzielczość i gęstość upakowania pikseli na ekranie komputera, na którym uruchamiana jest emulacja (**rysunek 15**).

Emulator może zawierać wszystkie przyciski, jakie znaleźć można w danym urządzeniu i teoretycznie powinien pozwalać udawać zjawiska zewnętrzne, oddziałujące na urządzenie z Androidem, dlatego oprócz samego ekranu, w zależności od symulowanego urządzenia, w oknie emulatora znaleźć można też dodatkowe przyciski (**rysunek 16**). Niestety, emulator dostępny w ramach SDK nie jest idealny, a do tego wolno działa. Dlatego na koniec tej części kursu pokażemy alternatywny emulator, który jest wyraźnie szybszy od standardowego i bardziej nam przypadł do gustu. Jego instalacja jest opcjonalna.

Rysunek 16. Emulowane urządzenie z rozbudowaną klawiaturą sprzętową

Tabela 4. Oprogramowanie zainstalowane w tej części kursu. Łącznie zajmują one około 21 GB przestrzeni dyskowej

Program	Wersja	Opis
NodeJS	0.10.35 (x64)	Środowisko wykonawcze dla aplikacji JavaScript
Git	1.9.5-preview20141217	Klient repozytoriów GIT
Cordova	4.1.2	Platforma programistyczna dla pisania aplikacji na systemy mobilne w językach HTML + JavaScript + CSS
Java SDK	8 update 25 (x64)	Narzędzia deweloperskie Java + środowisko uruchomieniowe Java (JRE)
ANT	1.9.4	Narzędzie do kompilowania, wykorzystywane w wybranej wersji platformy Cordova
Android Studio	Bundle 135.1641136	Zintegrowane, graficzne środowisko deweloperskie dla Androida, wraz z narzędziami deweloperskimi Android SDK i standardowym emulatorem
Genymotion	2.3.1	Szybki emulator Androida

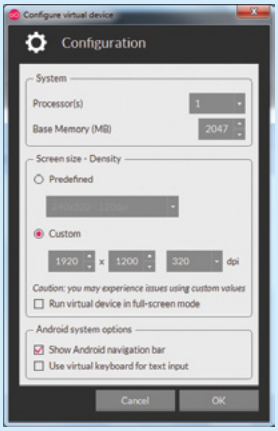


Rysunek 17. Przykładowe profile realnych urządzeń, możliwych do symulacji w emulatorze Genymotion

Genymotion

Emulator Genymotion pobieramy ze strony ***www.genymotion.com***. Jest on dostępny w wersji darmowej oraz płatnej (24,99 Euro za miesiąc). Wersja darmowa nie obsługuje wielu funkcji poza GPSem i kamerą oraz jest przeznaczona tylko do użytku osobistego. Wersja biznesowa wspiera obsługę wielodotyku, akcelerometry, symuluje IMEI, precyzyjnie, co do piksela odwzorowuje obraz na ekranie a nawet pozwala na symulację słabej jakości połączenia sieciowego.

Pobranie Genymotion wymaga rejestracji użytkownika, a ponieważ oprogramowanie bazuje na wirtualnym środowisku VirtualBox, są one domyślnie instalowane razem. Proces instalacji Genymotion jest bezproblemowy. Po uruchomieniu programu należy dodać nowe urządzenie wirtualne. Genymotion korzysta z internetowej bazy urządzeń, dzięki czemu można błyskawicznie uruchomić symulację na emulatorze, odpowiadającym jednemu z wielu popularnych, realnie istniejących smartfonów czy tabletów. Co więcej, przygotowane są nawet gotowe profile dla różnych wersji systemu zainstalowanego na danym modelu urządzenia (**rysunek 17**). Jeszcze przed uruchomieniem wybranego profilu urządzenia, można zmodyfikować jego szczegóły (**rysunek 18**), a gotowy, uruchomiony emulator ma dodatkowe przyciski (**rysunek 19**), służące np. do obracania ekranu, zmiany stopnia naładowania baterii czy choćby określenia pozycji odczytywanej z modułu GPS (**rysunek 20**).

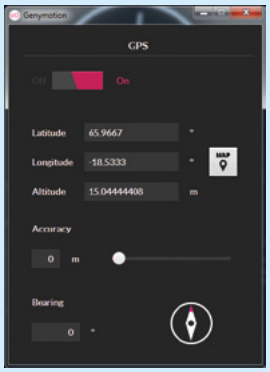


Rysunek 18. Opcje uruchamiania emulowanego urządzenia w programie Genymotion

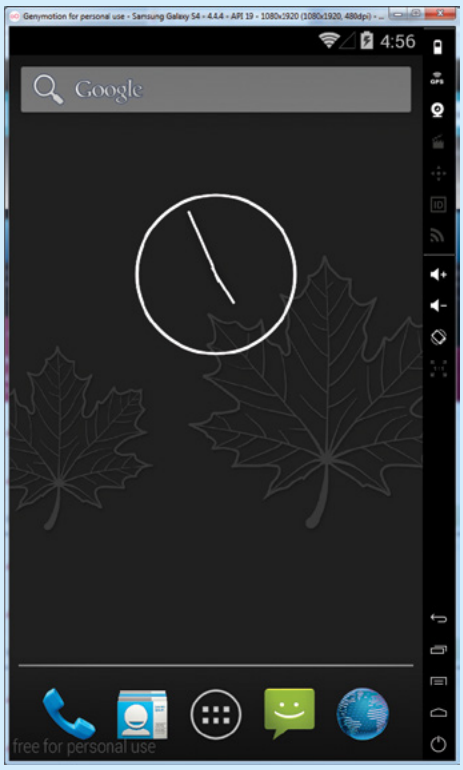
Podsumowanie

W niniejszej części kursu stworzyliśmy sobie kompletne środowisko deweloperskie dla aplikacji na Androida, przy czym niewiele potrzeba, by rozbudować je o programowanie aplikacji dla innych mobilnych systemów operacyjnych. Osoby umiejące programować strony internetowe już teraz mogą zacząć w bardzo łatwy sposób tworzyć i testować aplikacje androidowe. W następnych częściach kursu pokażemy, jak skorzystać z zewnętrznych bibliotek, z zasobów sprzętowych smartfona, jak komunikować się ze światem zewnętrznym z poziomu aplikacji oraz jak uzyskać dostęp do danych zapisanych w urządzeniu. Pokażemy też, jak skompilować aplikacje na inne systemy operacyjne i jak wykorzystać dostępne narzędzia programistyczne do usprawnienia programowania. Poświęcimy też trochę miejsca na przedstawienie dobrych praktyk programowania aplikacji z użyciem platformy Apache Cordova.

Marcin Karbowniczek, EP



Rysunek 20. Ustawienia emulowanego modułu GPS w Genymotion



Rysunek 19. Emulowane urządzenie androidowe w Genymotion, wraz z dodatkowymi przyciskami