AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Informatyki, Elektroniki i Telekomunikacji

Katedra Telekomunikacji

PRACA INŻYNIERSKA

Temat: Aplikacja sterująca odtwarzaczem multimedialnym dla urządzeń mobilnych o ograniczonych zasobach sprzętowych

Remote media controller for mobile devices with limited resources

Imię i nazwisko: Maciej Stankiewicz
Kierunek studiów: Elektronika i Telekomunikacja

Opiekun pracy: dr inż. Jarosław Bułat

Kraków, rok akadem. 2013/2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszy projekt inżynierski wykonałem(-am) osobiście i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Kraków, dnia ……. Podpis dyplomanta …….

Spis treści

Wstęp

Rozwój techniki wraz z postępującą miniaturyzacją spowodowały wzrost zainteresowania urządzeniami popularnie nazywanymi *wearable technology*, czyli elektroniki noszonej na ciele. Ta gałąź elektroniki zdecydowanie rośnie w siłę, wobec czego pojawia się co raz więcej sprzętu ułatwiającego nam codzienne życie, będąc ciągle przy nas. Ponadto postęp tego typu urządzeń wraz z rozpowszechnieniem *smartfonów* pozwolił również rozwinąć systemy na platformy mobilne.

Dlatego też, w niniejszej pracy podjęto się stworzenia aplikacji sterującej odtwarzaczem multimedialnym dla nowoczesnych zegarków – *smartwatchy*, które są właśnie przedstawicielami elektroniki będącej częścią naszego ubioru. Program jest dedykowany na obecnie najpopularniejszy mobilny system operacyjny Android i wraz z odtwarzaczem multimedialnym wchodzi w skład systemu multimedialnego. Pierwsza z aplikacji przeznaczona na zegarki elektroniczne pozwoli użytkownikowi kontrolować drugi program, który również będzie zaimplementowany na tym samym systemie operacyjnym. Jego zadaniem będzie strumieniowanie plików audio z zewnętrznego serwera.

W pierwszym rozdziale zawarto opis systemu wraz z jego schematem działania i obsługą użytkownika. Dokonano również analizy rynku pod względem istniejących już urządzeń i systemów operacyjnych, jak i możliwości rozwoju *smartwatchy*. Rozdział drugi poświęcony jest implementacji systemu. W ostatnim rozdziale przedstawiono wyniki testów wydajnościowych aplikacji wraz z prezentacją możliwości zegarków elektronicznych.

1. Cel i założenia projektu

Na cel niniejszej pracy składają się dwa zadania. Pierwszym z nich jest zaprojektowanie oraz zaimplementowanie aplikacji umożliwiającej sterowanie odtwarzaczem multimedialnym, którego docelową platformą będzie *smartwatch* posiadający system operacyjny Android. Podstawowe funkcje programu kontrolującego są następujące: organizacja tzw.: *playlist*, stop, odtwarzaj, następny, poprzedni, głośniej, ciszej. Drugim zadaniem jest stworzenie odtwarzacza multimedialnego, który będzie przeznaczony dla urządzeń mających możliwość podłączenia do wzmacniacza audio bądź też telewizora. Urządzeniem wykorzystanym w opisywanym projekcie jest tzw. „*Android dongle*", czyli miniaturowe urządzenie z zainstalowanym systemem Android oraz posiadające wyjście HDMI. Program działający na tym urządzeniu będzie odpowiedzialny za odtwarzanie utworów wybranych przez użytkownika. Docelowym źródłem danych będzie serwer. Aplikacja do poprawnego działania wymaga nieustannego połączenia z siecią domową. Natomiast komunikacja pomiędzy aplikacjami będzie możliwa za pomocą technologii bezprzewodowej Bluetooth.

1.1. Środowiska sprzętowe

W tym podrozdziale przedstawione będą urządzenia wykorzystane w niniejszym projekcie inżynierskim. Zostanie opisane pojęcie *„wearable technology”*, a w szczególności nowoczesne inteligentne zegarki elektroniczne. Ponadto zostanie zaprezentowana część stacjonarna systemu, czyli *Android dongle* wraz z wyjaśnieniem, dlaczego akurat to urządzenie zostało wybrane.

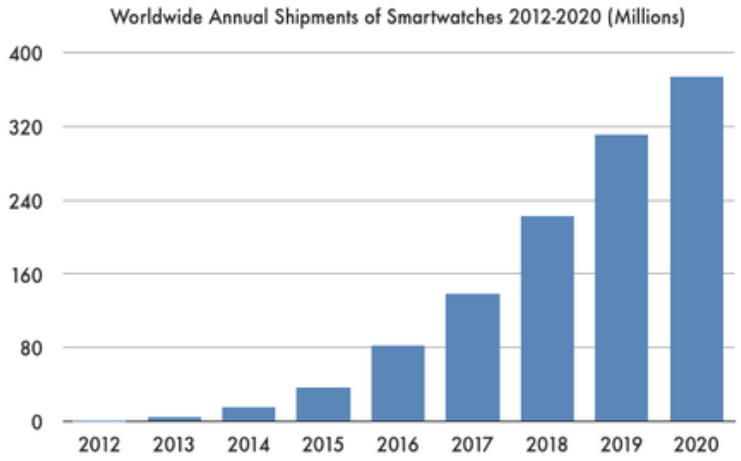
URZĄDZENIE MOBILNE TYPU SMARTWATCH

Smartwatch, inaczej „inteligentny zegarek”, jest przedstawicielem *wearable technology*, czyli elektroniką, którą można nosić na ciele w czasie normalnego funkcjonowania. Obecnie te urządzenia są jedynie gadżetami mającymi na celu umożliwić jeszcze efektywniejsze używanie *smartfonów*. Dostępne dzisiaj zegarki pozwalają jedynie odbierać połączenia, sms-y i emaile. Można rzec, że są po prostu dodatkowym wyświetlaczem dla telefonu, wobec czego bez niego są bezużyteczne. Jednakże w przyszłości być może właśnie dzięki zegarkom użytkownicy będą mogli wykonywać połączenia, wysyłać wiadomości tekstowe, robić zdjęcia, czy też korzystać z Internetu oraz nawigacji GPS.

Pojęcie *smartwatch* nie jest niczym nowym. Sama idea takich skomputeryzowanych urządzeń sięga lat 80, gdyż wtedy były już dostępne zegarki z pamięcią elektroniczną. W roku 2000 IBM opracował *smartwatch* pracujący pod kontrolą Linuxa. W następnych latach Microsoft mógł pochwalić się swoim inteligentnym zegarkiem nazwanym „SPOT watch”. Do walki o ten segment rynku pod koniec pierwszej dekady XXI wieku włączyli się również producenci telefonów komórkowych, mianowicie LG z produktem GD910 oraz Samsung z S9110. Jednakże obydwa urządzenia mimo wielu zalet oferowały za mało, dlatego też słabo się sprzedawały. Prawdziwy „boom” na tego typu sprzęt miał miejsce w roku 2012, dzięki zegarkowi Pebble zaprezentowanemu poprzez *crowdfoundingowy* serwis Kickstarter. Produkt ten został sfinansowany przez społeczność nim zainteresowaną w kwocie przekraczającej 10 milionów dolarów. Po sukcesie tej kampanii *smartfonowi* potentaci Sony oraz Samsung zaprezentowali swoje *smartwatche* działające już pod kontrolą systemu Android. Adekwatne do obecnej sytuacji na rynku będą słowa analityka

Marshal'a Cohen z NPD Group w Nowym Jorku: „Obserwujemy wyścig, na mecie którego czeka coś, co możesz po prostu ubrać. Z każdej strony producenci atakują projektami, pomysłami, domysłami. Próbują na nowo odkryć telefon. Zresztą, nie tylko firm technologicznych to dotyczy, bo swój udział mają również wytwórcy akcesoriów czy nawet projektanci mody."

Biorąc pod uwagę powyższe rozważania oraz raport wykonany przez firmę NextMarket Insights[4], która spróbowała określić, jak będzie się rozrastać rynek zegarków elektronicznych, można śmiało stwierdzić, iż zainteresowanie tym mobilnymi urządzeniami będzie z roku na rok stopniowo wzrastać, gdyż zwiększy się zapotrzebowanie na tego typu elektronikę. Prognozę rocznej sprzedaży zaprezentowano na rysunku 1. W roku 2014 ma wynieść 15.4 miliona urządzeń, by w 2020 osiągnąć blisko 374 miliony. W tej analizie pod uwagę wzięto kilka elementów, które mają mieć wpływ na tak ogromny wzrost. Czynnikami tymi są: zastąpienie tradycyjnych zegarków *smartwatchami*, rozwój i dostosowanie mobilnych systemów operacyjnych do tych urządzeń wraz ze wzrostem ilości dostępnych aplikacji oraz różnorodność pod względem mody i trybu życia. W efekcie firma NextMarket Insights przewiduje, iż inteligentne zegarki nie będą już tylko akcesorium, gadżetem, czy też dodatkowym wyświetlaczem do *smartfonu*, ale staną się nim samym. Dzięki czemu za pomocą zegarka będziemy mogli wykonywać połączenia, korzystać z nawigacji GPS oraz Internetu (transmisja danych).



Rysunek 1. Analiza rocznej sprzedaży *smartwatchy* w latach 2012-2020

Źródło: Smartwatch forecast 2013-2020 [4]

Obecnie na rynku dostępnych jest kilkanaście inteligentnych zegarków. Część z nich została wyprodukowana przez elektronicznych potentatów, a niektóre zostały sfinansowane przez społeczność za pomocą serwisu Kickstarter. Niniejsze *smartwatche* pracują pod różnymi systemami operacyjnymi, z czego kilka urządzeń korzysta z Androida, bądź jego modyfikacji. Do najpopularniejszych z nich należą Sony Smartwatch 2, Samsung Galaxy Gear, i'm Watch, Neptune Pine oraz Omate Truesmart. W tabeli 1 zaprezentowano szczegółową specyfikację tych modeli.

Z porównania zaprezentowanego w tabeli 1. wynika, że obecnie dostępne zegarki są stosunkowo zróżnicowane ze względu na swe specyfikacje. Można zauważyć, że wszystkie wyświetlacze, z wyjątkiem Neptune Pine, mają wielkość około 1.6”, natomiast mając na uwadze ich rozdzielczość średnio jest to 240 na 240 pikseli. Jednakże w projektowanej aplikacji zostanie wykorzystana rozdzielczość 320 na 320, czyli ta zaproponowana przez Samsunga. Ponadto cechą wspólną tychże urządzeń jest moduł do bezprzewodowej komunikacji Bluetooth. Dlatego też w niniejszym projekcie wykorzystano właśnie tą technologię. *Smartwatche* znacząco różnią się ze względu na użyte w nich procesory, które są zarówno jednordzeniowe o taktowaniu niższym niż 200 MHz, ale i również dwurdzeniowe o częstotliwości przekraczającej 1 GHz. Biorąc pod uwagę dostępną pamięć operacyjną RAM można uznać, że docelowo urządzenia będą miały 512 MB. Niektóre zegarki już mogą korzystać z sieci komórkowej, WiFi oraz odbiorników GPS. Jednakże urządzenia te są stosunkowo drogie, gdyż poza Sony Smartwatch 2, ich cena oscyluje wokół 300 dolarów. Dlatego aby ta gałąź elektroniki odniosła pożądany sukces zegarki muszą stanieć, ale przede wszystkim producenci muszą się skoncentrować na wydłużeniu czasu pracy baterii.

Tabela 1. Porównanie specyfikacji istniejących *smartwatchy*

	Sony Smartwatch 2	Samsung Galaxy Gear	i'm Watch	Neptun Pine	Omate Truesmart
Procesor	ARM Cortex-M4, 1 rdzeń 180 MHz	ARM, 1 rdzeń 800 MHz	IMX233	Qualcomm Snapdragon S4, 2 rdzenie 1,2 GHz	Cortex A7, 2 rdzenie 1,3 GHz
Pamięć	Brak danych	512 MB RAM + 4 GB	128 MB RAM + 4 GB	512 MB RAM + 16/32 GB	512 MB RAM + 4 GB lub 1 GB RAM + 8 GB
Wyświetlacz	1,6”	1,63”	1,56”	2,41”	1,54”
Rozdzielczość	220x176	320x320	240x240	320x240	240x240
Bluetooth	Tak, v.3.0	Tak, v.4.0	Tak	Tak, v.4.0	Tak, v.4.0
Sieć komórkowa	Brak	Brak	Brak	Tak	Tak
Bateria	3-4 dni	1 dzień	1-4 dni	1-5 dni	do 4dni
Mikrofon	Brak	Tak	Tak	Tak	Tak
Głośnik	Brak	Tak	Tak	Tak	Tak
Kamera	Brak	1.9 MPx	Nie	5.0 MPx	5.0 MPx
System operacyjny	System kompatybilny z Android 4.0+	Modyfikacja Androida	i'm Droid 2 (modyfikacja Androida)	Android 4.1.2	Omate UI 1.0 (modyfikacja Androida 4.2.2)
Wodoodporny	Tak	Tak	Nie	Tak	Tak
Cena	170-199$	299$	349$	335$ (16 GB) Lub 395$ (32 GB)	249$ (512 MB RAM +4GB) lub 299$ (1 GB RAM + 8 GB)
Inne	NFC			Micro-SIM, GPS, WIFI	Micro-SIM, WIFI, GPS, MicroSD

ANDROID DONGLE

Android jest obecnie najpopularniejszym systemem dla urządzeń mobilnych. Jednakże za sprawą co raz popularniejszych telewizorów typu *SmartTV*, Android został przeniesiony również na ekrany odbiorników TV, bynajmniej nie dzięki producentom tychże urządzeń. Pojęcie *SmartTV* oznacza, iż telewizor może z powodzeniem pełnić funkcje domowego centrum rozrywki wraz z dostępem do Internetu. Niemniej jednak nie wszyscy posiadają tego typu odbiorniki oraz nie wszyscy mogą sobie pozwolić na wydatek minimum 1500 zł na zakup nowego urządzenia. Z pomocą dla tych osób wychodzą naprzeciw producenci oferujący małe przystawki montowane do telewizora za pomocą portu HDMI, zbudowane na systemie Android oraz rozszerzające możliwości tradycyjnego odbiornika o funkcje oferowane przez *SmartTV*. Korzystna jest również cena takich urządzeń rozpoczynająca się już od 150 zł. Dodatkowym atutem tych przystawek jest sklep z aplikacjami wykorzystywany na Androidzie, czyli Google Play, który oferuje wielokrotnie więcej programów w stosunku do tych dostępnych dla użytkowników klasycznych *SmartTV* i niewiele wskazuje, aby to miało się zmienić. Ponadto urządzenia te charakteryzują się bardzo dobrymi parametrami technicznymi.

Tabela 2. Porównanie specyfikacji wybranych modeli przystawek *Android dongle*

	Measy U2B	Measy U2C	Measy U4B	Cabletech RK3066	Cabletech URZ0351
Procesor	Cortex A9, 2 rdzenie 1,6 GHz	Cortex A9, 2 rdzenie 1,6 GHz	Cortex A9, 4 rdzenie 1,6 GHz	Cortex A9, 2 rdzenie 1,5 GHz	Cortex A9, 2 rdzenie 1,6 GHz
RAM	1 GB	1 GB	2 GB	1 GB	1 GB
Pamięć	8 GB	8 GB	8 GB	4 GB	4 GB
WiFi	Tak	Tak	Tak	Tak	Tak
Bluetooth	Tak	Tak, v.4.0	Tak, v.4.0	Nie	Tak
Mikrofon	Nie	Tak	Nie	Nie	Nie
Kamera	Nie	Tak	Nie	Nie	Nie
USB	1	1	1	1	2
Gniazdo kart SD	Tak	Tak	Tak	Tak	Tak
System operacyjny	Android 4.1	Android 4.1	Android 4.2	Android 4.1	Android 4.1
Cena	ok. 230 zł	ok. 270 zł	ok. 390 zł	ok. 200 zł	ok. 230 zł

W tabeli 2 przedstawiono obecnie najlepiej dostępne produkty typu *Android dongle*. Jak widać są to urządzenia mające minimum dwurdzeniowy procesor o częstotliwości taktowania ponad 1.5 GHz oraz minimum 1 GB RAM i 4 GB dostępnej pamięci. Każda z tych przystawek ma możliwość komunikacji za pomocą WiFi, wbudowane ma gniazdo kart pamięci microSD oraz przynajmniej 1 port USB, który pozwala nam podłączyć zewnętrzny dysk, bądź też urządzenie sterujące – mysz lub kontroler. Ponadto zainstalowanym systemem operacyjnym jest Android w wersji 4.1 lub nowszej. Wszystkie urządzenia cechuje niski pobór energii, gdyż są zasilane poprzez port microUSB (DC 5V/2A). Różnice pojawiają się dopiero ze względu na dodatkowe parametry, gdyż nie wszystkie mają wbudowany moduł Bluetooth, a tylko jedno z tych urządzeń, mianowicie Measy U2C, posiada wbudowaną kamerę wraz z mikrofonem, którą można wykorzystać do video-rozmów. Ponadto wymienione produkty różnią się ze względu na cenę od 200 do prawie 400 zł, ale wynika to z zastosowanych różnych podzespołów. Dlatego biorąc pod uwagę powyższe urządzenia najrozsądniejszym zakupem będzie Measy U2B, bądź też U2C, jeśli mamy zamiar wykorzystać kamerę.

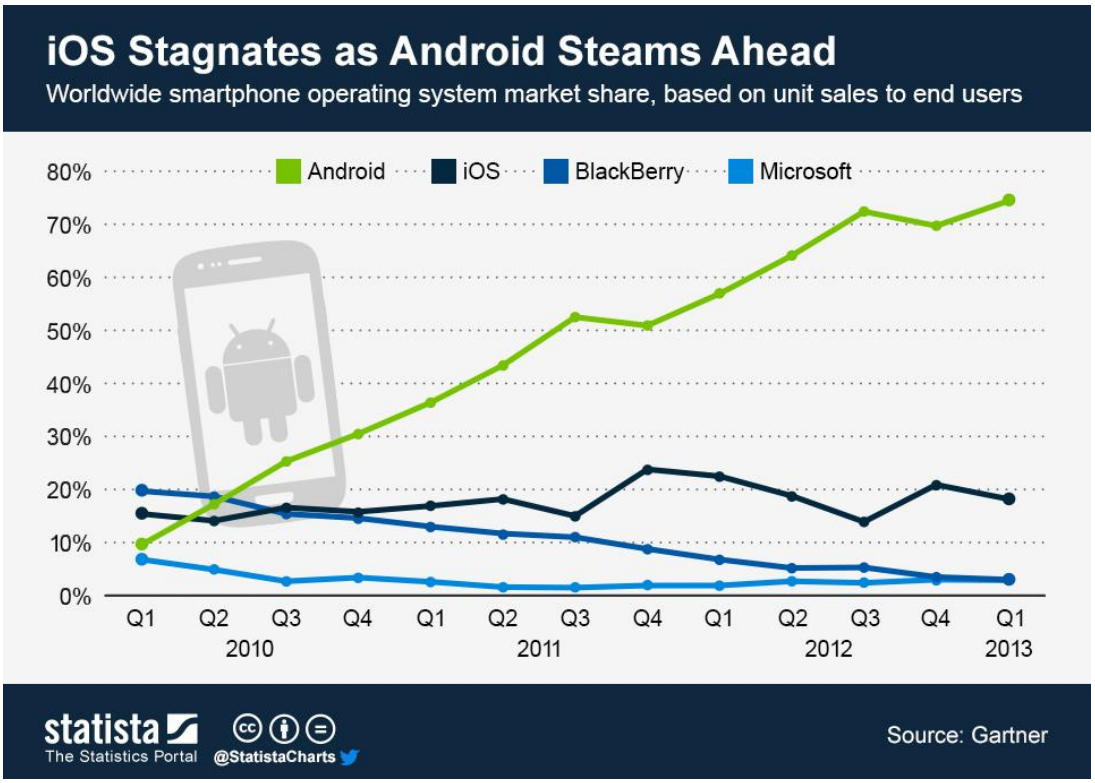
Dlaczego właśnie tego typu urządzenie zostało wybrane do systemu opisywanego w niniejszej pracy? Otóż odpowiedź jest bardzo prosta, mianowicie sprzęt ten posiada dobre parametry techniczne, niski pobór mocy, dostęp do sieci internetowej, moduł bezprzewodowej komunikacji Bluetooth oraz możliwość podłączenia do telewizora za pomocą HDMI, bądź też wzmacniacza audio, jeśli sygnał z tego portu zostanie podzielony na obraz i dźwięk. Ponadto takie urządzenie korzysta z takiego samego systemu operacyjnego jak aplikacja, która będzie sterowała odtwarzaczem, czyli Androida oraz cechuje się stosunkowo niską ceną. Do testów jako odtwarzacza multimedialnego wykorzystano produkt Measy U2C.

1.2. Środowiska programowe

W tym podrozdziale zostanie zaprezentowany system operacyjny zainstalowany na wykorzystywanych urządzeniach (Android) oraz środowisko programistyczne dla tejże platformy - Eclipse. Czytelnik będzie miał możliwość również zapoznać się z technologiami wykorzystanymi w opisywanych aplikacjach.

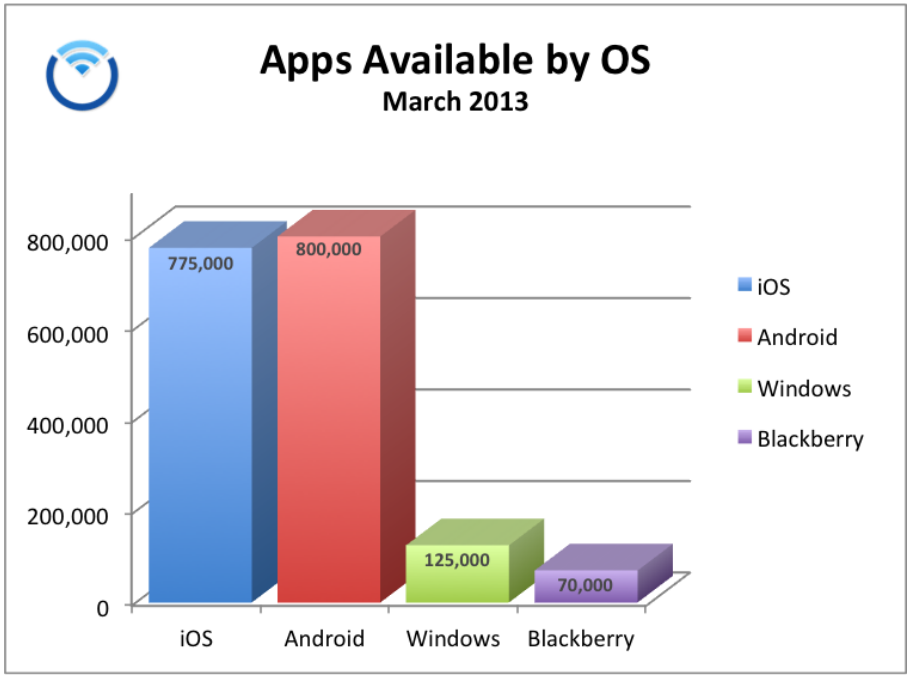
SYSTEM OPERACYJNY ANDROID

Android został zaprojektowany przez małą firmę Android Inc., którą w roku 2005 wykupiło Google. W 2007 roku z inicjatywy Google powstał sojusz Open Handset Alliance (OHA) zrzeszający firmy związane z telekomunikacją. Skutkiem założenia tej organizacji było udostępnienie wczesnej wersji tego systemu wraz z SDK (zestawem narzędzi dla programistów), co spowodowało możliwość pisania aplikacji na Androida przez niezależnych developerów. We wrześniu 2008 roku została wprowadzona pierwsza komercyjna wersja tego systemu, natomiast miesiąc później został wprowadzony na rynek pierwszy telefon wykorzystujący Androida – HTC Dream, znany także jak T-Mobile G1. System ten stosunkowo szybko zdobywał uznanie wśród użytkowników, co zaowocowało co raz większą ilością dostępnych urządzeń pracujących pod jego kontrolą. Nie bez znaczenia było zainteresowanie programistów, dzięki którym było co raz więcej aplikacji możliwych do pobrania ze sklepu Google Play. Sukcesywnie wprowadzane dalsze poprawki spowodowały, że Android stał się obecnie najpopularniejszym systemem mobilnym, a urządzenia działające pod jego kontrolą sprzedają się najlepiej na świecie nieprzerwanie od trzeciego kwartału 2010 roku (rysunek 2). Z przedstawionych danych wynika, że blisko 80% sprzedawanych urządzeń na rynku posiada zainstalowany system Android. W dużym stopniu spowodowane jest to niewielkimi opłatami licencyjnymi za umieszczenie tego systemu w swoim sprzęcie.



Rysunek 2. Sprzedaż *smartfonów*

Źródło: http://www.statista.com/chart/1099/smartphone-operating-system-market-share/



Rysunek 3. Wykres z ilością aplikacji w poszczególnym markecie

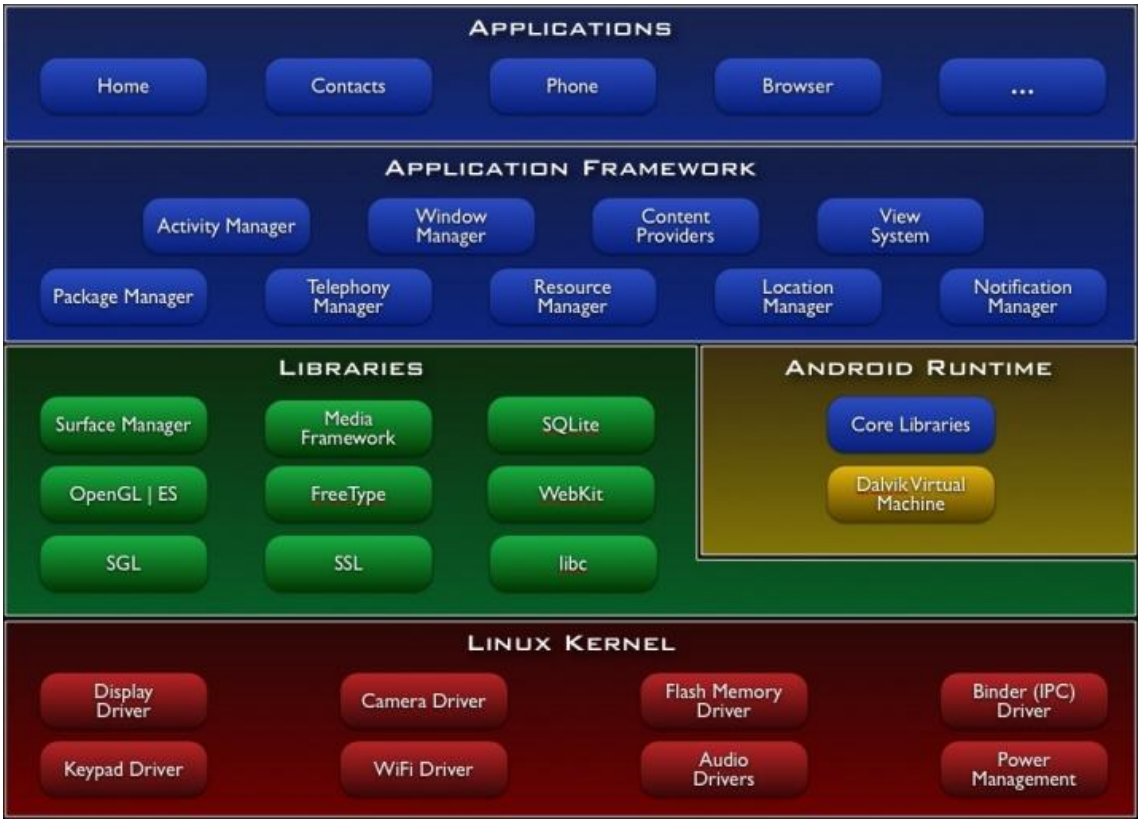
Źródło: http://www.pureoxygenmobile.com/how-many-apps-in-each-app-store/

Biorąc pod uwagę ilość dostępnych aplikacji dla wiodących mobilnych systemów operacyjnych, tj. iOS, Android, Windows oraz Blackbery, również prym wiedzie sklep Google Play co zaprezentowano na rysunku 3. Ilość programów dla urządzeń z Androidem przekroczyła już 800 tysięcy, co spowodowało zastąpienie dotychczasowego lidera - AppStore. Jednakże wpływ na taki przebieg zdarzeń również ma restrykcyjna polityka wprowadzania nowych aplikacji stosowana przez Apple'a.

Do tej pory przedstawiono historię oraz pozycję Androida na rynku systemów mobilnych, dlatego też poniżej zostanie zaprezentowana architektura tego systemu. Jak widać na rysunku 4 stos Androida został podzielony na 5 warstw, zaczynając od najniższej są to:

- **Jądro Linuksa**, które stanowi rdzeń platformy oraz jest odpowiedzialne za sterowniki do sprzętu, dostęp do zasobów i zarządzanie energią. Obecnie wykorzystywane jest jądro w wersji 3.0 – od Androida 4.0.
- **Biblioteki natywne** napisane głównie w C/C++. W tej warstwie umieszczono biblioteki tj. OpenGL (generowanie grafiki 3D), SGL (wyświetlanie grafiki 2D), WebKit (obsługa przeglądarki internetowej), SSL (szyfrowanie przesyłanych danych), FreeType (generowanie i renderowanie fontów), biblioteka wykonawcza języka C (libc), SQLite (obsługa baz danych SQL), Surface Manager (tworzenie okien na ekranie) oraz Media Framework (odtwarzanie i nagrywanie różnych formatów obrazu, audio i wideo).
- **Środowisko wykonawcze (Android Runtime)** jest umieszczone na tym samym poziomie, co warstwa bibliotek. Tutaj zawarte są podstawowe biblioteki Java, ponieważ aplikacje tworzenie na ten system są pisane w tym języku programowania. Ponadto, a może przede wszystkim, w tej warstwie zawarta jest maszyna wirtualna **Dalvik** będąca jednym z kluczowych elementów tego systemu. Dalvik VM jest maszyną wirtualną dedykowaną dla systemu Android zdecydowano się nie korzystać z tradycyjnej głównie z powodów licencyjnych (patenty) oraz z powodu ograniczonych zasobów pamięci oraz procesora. Dlatego też Dalvik VM jest zoptymalizowany do niskich wymagań dotyczących tych zasobów oraz umożliwia uruchomienie wielu instancji wirtualnych maszyn naraz (to właśnie w nim uruchomiane są wszystkie aplikacje w tym systemie).

- **Framework aplikacji** jest warstwą, w której umieszczone zostały klasy służące do tworzenia aplikacji na Androida. Umożliwia m.in. zarządzanie zasobami urządzenia (Resource Manager), połączeniami głosowymi (Telephony Manager), wymianą danych pomiędzy aplikacjami (Content Providers), czy też lokalizacją urządzenia (Location Manager).
- **Aplikacja** stanowi najwyższą warstwę stosu Androida. Jak sama nazwa wskazuje wszystkie aplikacje, standardowe, czy też pobrane z zewnętrznych źródeł działają właśnie w tej warstwie. Jest ona wykorzystywana głównej mierze przez przeciętnego użytkownika urządzeń z zainstalowanym systemem operacyjnym Android.



Rysunek 4. Architektura systemu Android

Źródło: http://elinux.org/Android_Architecture

Przed rozpoczęciem pisania programów na system Android deweloperzy bezwzględnie muszą zaznajomić się z podstawowymi pojęciami szkieletu aplikacji, takimi jak: widok, aktywność, intencja, dostawca treści oraz usługa. Widoki to elementy interfejsu użytkownika, dlatego mogą przybierać kształt przycisku, etykiety, pola tekstowego, czy też innych składników. Aktywność jest interfejsem użytkownika,

dlatego składa się z przynajmniej jednego widoku. Przeważnie obrazuje pojedynczy ekran aplikacji. Intencja oznacza zamiar wykonania jakiegoś zadania. Jest wykorzystywana np. do rozpoczęcia aktywności, uruchomienia usługi, wybierania lub odbierania połączenia telefonicznego, bądź też nadania komunikatu. Dostawca treści pozwala korzystać jednej aplikacji z danych innej, dzięki czemu możliwe jest współdzielenie informacji pomiędzy programami. Ostatnim z pojęć jest usługa, która oznacza procesy działające w tle i dzieli się na dwa rodzaje: usługi lokalne (dostępne wyłącznie dla konkretnej aplikacji) oraz zdalne (dostępne dla wielu aplikacji).

ECLIPSE

Eclipse jest zalecanym środowiskiem programistycznym umożliwiającym tworzenie oprogramowania na system operacyjny Android. Narzędzie to zostało stworzone przez firmę IBM, a następnie udostępnione społeczności Open Source. Głównym atutem tego programu jest elastyczność i możliwość dodawania modułów do różnych języków programowania. Do zalet korzystania z Eclipse można zaliczyć także „kolorowanie składni", weryfikację błędów, autouzupełnienie kodu, zautomatyzowane importowanie odpowiednich klas czy też funkcje refaktoryzacji kodu. Dzięki odpowiednim wtyczkom dostarczonym przez Google środowisko Eclipse pozwala w prosty sposób uruchomić i testować napisane aplikacje na odpowiednim urządzeniu bądź też emulatorze. Ponadto ułatwia proces debugowania tworzonych programów dzięki narzędziu Logcat, które musi być uruchomione na urządzeniu i pozwala na zbieranie dzienników systemu oraz działających na nim aplikacji.

WYKORZYSTANE TECHNOLOGIE

W systemie opisywanym w tej pracy wykorzystano następujące technologie: język programowania Java, język znaczników XML, protokół HTTP oraz standard Bluetooth.

Java jest językiem programowania obiektowego stworzonym przez firmę Sun Microsystems. Programy w nim tworzone kompilowane są do kodu bajtowego, który jest wykonywany przez maszynę wirtualną (w przypadku Androida przez Dalvik VM), dzięki czemu jest niezależny od systemu operacyjnego oraz architektury. W prezentowanej pracy jest wykorzystany do implementacji obydwu aplikacji.

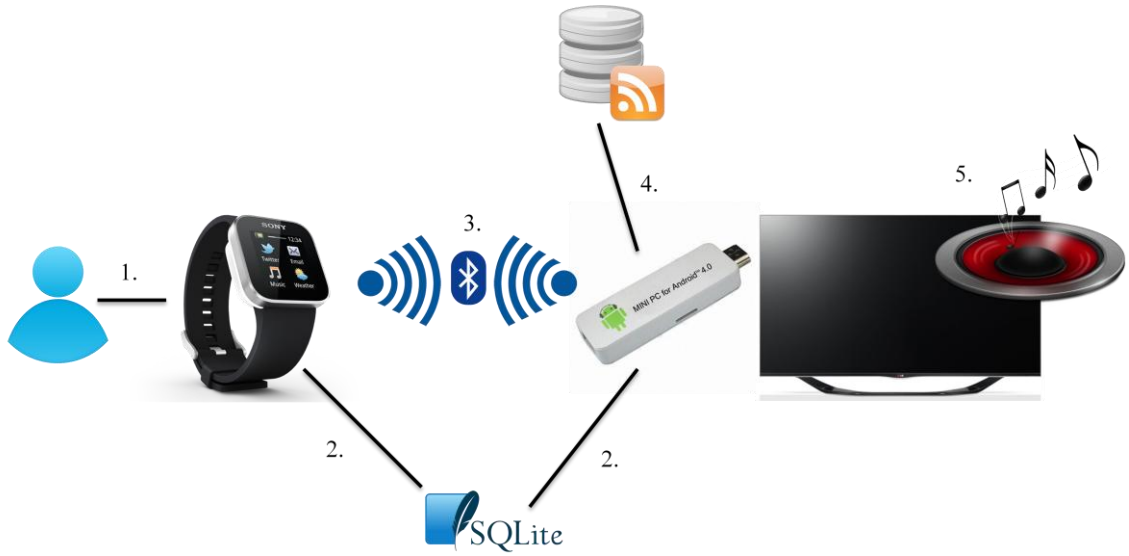
XML, jest niezależnym od platformy, językiem znaczników umożliwiającym łatwą wymianę dokumentów pomiędzy różnymi systemami. W opisywanym projekcie znajduje dwojakie zastosowanie. Po pierwsze jest odpowiedzialny za zasoby oraz budowę interfejsu użytkownika w aplikacjach na Androida. Natomiast jego drugim zastosowaniem jest plik konfiguracyjny napisany w tym języku z wszystkimi utworami znajdujący się na serwerze. Ów plik po odpowiednim przetworzeniu (analizie) umożliwia stworzenie bazy danych wykorzystywanej w obydwu programach.

Protokół HTTP jest protokołem pozwalającym na komunikację serwerów WWW z klientami i właśnie takie zadanie pełni w opisywanym systemie, a dokładniej rzecz ujmując odpowiada za strumieniowanie utworów muzycznych dostępnych na zewnętrznym serwerze.

Bluetooth jest technologią bezprzewodowej komunikacji krótkiego zasięgu korzystającą z fal radiowych o częstotliwości 2.4 GHz umożliwiającą komunikację pomiędzy różnymi urządzeniami elektronicznymi. System został zaprojektowany przez firmę Ericsson w 1994 roku, natomiast na rynku pojawił się w 1999 roku. Jego teoretyczny zasięg wynosi do 100m, jednakże najpowszechniejsze w użyciu są urządzenia umożliwiające komunikację do 10 m. Maksymalny transfer przesyłania danych jest zależny od wersji Bluetooth (może on wynosić do 5 MB/s). W opisywanym systemie umożliwia połączenie pomiędzy *smartwatchem* (aplikacja sterująca) a przystawką *Android dongle* (odtwarzacz multimedialny).

1.3. Schemat działania systemu

Znając już podstawową funkcjonalność, dostępną platformę sprzętową oraz programową ten podrozdział ma za zadanie umożliwić czytelnikowi poznanie bardziej szczegółowych informacji dotyczących działania opisywanego systemu.



Rysunek 5. Schemat obsługi systemu

Na działanie niniejszego systemu składają się następujące elementy przedstawione na rysunku 5:

1. Obsługa użytkownika.
2. Zapis danych do lokalnej bazy danych SQLite.
3. Komunikacja pomiędzy aplikacjami za pomocą Bluetooth.
4. Strumieniowanie danych z serwera.
5. Odtwarzanie audio.

Użytkownik ma możliwość w jak najprostszy sposób kontrolować odtwarzacz. Nikt nie ma przy sobie cały czas *smartfonu*, wobec czego *smartwatch* może być dobrą alternatywą pilota do odtwarzacza audio. Osoba korzystająca z zegarka może bezproblemowo zmieniać aktualny utwór, w dowolnym momencie zatrzymać, bądź też

wznowić odtwarzanie, jak i wybrać *playlistę*. Do dyspozycji użytkownika będzie również kontrola głośności.

Programy użyte w przedstawionym systemie korzystają z lokalnej bazy danych SQLite, aby mieć dostępne wszystkie utwory oraz *playlisty*. Dane dostarczone z serwera trafiają do odpowiednich plików bazodanowych i zostają w nich zapisane w pamięci wewnętrznej urządzenia. Po wyłączeniu lub zatrzymaniu aplikacji, jak i po zresetowaniu systemu dane te pozostają niezmienione. Ich zmiana następuje tylko w przypadku zmiany na serwerze, natomiast trwałe usunięcie jest możliwe tylko wraz z odinstalowaniem programów.

Obydwie aplikacje, jak już wcześniej było wspomniane, komunikują się ze sobą za pomocą technologii Bluetooth. Zestawienie połączenia rozpoczyna program dostępny na *smartwatchu*, wobec czego odtwarzacz multimedialny może działać niezależnie od aplikacji sterującej. Gdy urządzenia są już ze sobą połączone, następuje wymiana danych pomiędzy nimi, która rozpoczyna się od sprawdzenia aktualnej wersji bazy danych oraz jej ewentualnej wymiany. Następnie przesyłany jest numer aktualnie odtwarzanego utworu, dzięki czemu można również określić *playlistę*. Oczywiście zachodzi to tylko w przypadku, gdy w programie uruchomionym na przystawce *Android dongle* jest odtwarzany jakiś plik. W przeciwnym zaś razie aplikacja sterująca pozwoli, aby to użytkownik określił *playlistę* wraz z utworem. Komunikacja pomiędzy programami odbywa się na zasadzie chatu, w którym przesyłane są odpowiednie komendy, które zostaną zaprezentowane w dalszej części tej pracy.

Dane dostępne na serwerze są strumieniowane przez przystawkę *Android dongle*. Oznacza to, iż są odtwarzane w czasie rzeczywistym bez konieczności ich pobierania. Takie rozwiązanie ma swoje plusy, jak i minusy. Do pozytywnych aspektów należy przede wszystkim brak obciążenia pamięci masowej urządzenia oraz fakt, iż nie trzeba czekać na pobranie całego utworu. Ponadto dzięki zastosowaniu *streamingu* użytkownik może korzystać z tych samych danych umiejscowionych na serwerze za pomocą wielu urządzeń. Jak już zostało wspomniane takie rozwiązanie niesie też za sobą pewne niedogodności, a mianowicie przez cały czas odtwarzania trzeba mieć dostęp do sieci domowej, co nie zmienia faktu, iż zastosowanie strumieniowania jest dobrym wyborem.

Przystawka *Android dongle* została podłączona do telewizora, wobec czego odtwarzane utwory można usłyszeć właśnie dzięki temu urządzeniu. Jak już wcześniej zostało napisane odbiornik telewizyjny nie jest jedyną możliwością odtwarzania. Alternatywą dla niego może być wzmacniacz, bądź też dowolny zestaw głośników potrafiący odczytać strumień audio z interfejsu HDMI.

2. Implementacja

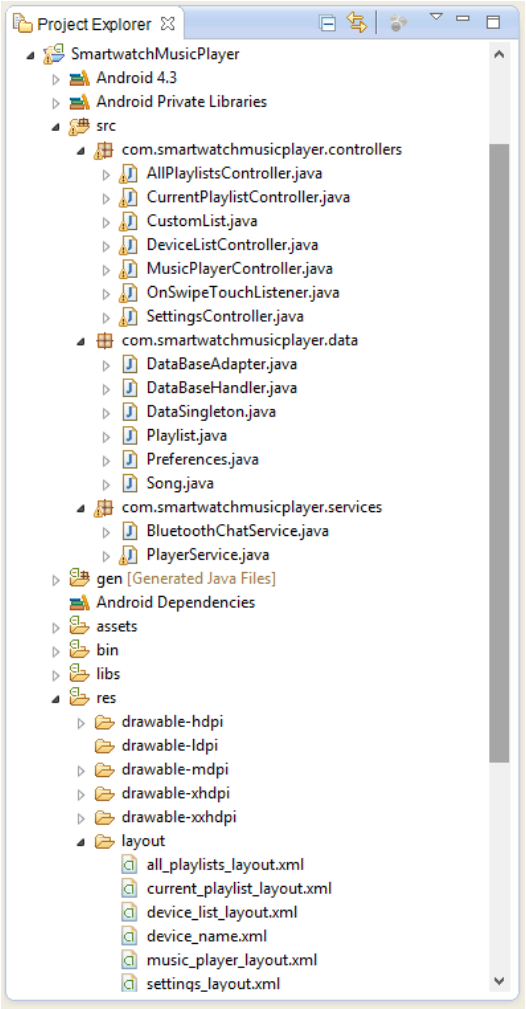
Rozdział ten poświęcony jest implementacji systemu. Składa się z dwóch podrozdziałów – pierwszy prezentuje sposób implementacji aplikacji sterującej, natomiast drugi odtwarzacza multimedialnego.

2.1. Aplikacja mobilna sterująca odtwarzaczem

W tym podrozdziale zostanie szczegółowo opisana struktura programu przeznaczonego na *smartwatch*, sposób komunikacji z odtwarzaczem multimedialnym oraz sposób tworzenia bazy danych. Ponadto zostanie zaprezentowany wygląd tej aplikacji oraz sposób użycia.

STRUKTURA APLIKACJI

Program został zaimplementowany na *smartwatche*, wobec czego jest przygotowany dla urządzeń z minimalną wersją Androida 4.0. Aplikacja zbudowana została zgodnie z wzorcem projektowym Model-Widok-Kontroler, dzięki czemu w łatwy sposób będzie mógł być rozbudowany w przyszłości. Aplikacja składa się z pięciu widoków które umieszczone są w katalogu res/layout (rysunek 6). Kontrolery za nie odpowiedzialne umieszczone są w pakiecie *com.smartwatchmusicplayer.controllers*, wobec czego możemy wyróżnić tutaj 5 klas, a mianowicie: *SettingsController*, *CurrentPlaylistController*, *MusicPlayerController*, *AllPlaylistsController* oraz *DeviceListController*. Do tego pakietu należy również klasa *OnSwipeTouchListener*, która odpowiedzialna jest za możliwości nawigowania pomiędzy widokami oraz znajduje szersze zastosowanie przy kontroli odtwarzacza. Bardziej szczegółowy opis będzie zaprezentowany w dalszej części pracy.



Rysunek 6. Struktura aplikacji

W pakiecie *com.smartwatchmusicplayer.data* znajdują się klasy odpowiedzialne za dane przetwarzane w aplikacji, które możemy podzielić na dwie grupy. Do pierwszej z nich można zaliczyć *Preferences*, która odpowiada za zapis opcji, do których należą odtwarzanie losowe lub zapętlenie, wówczas gdy program zostanie wyłączony, bądź zatrzymany oraz *DataSingleton*, która również jest wzorcem projektowym ograniczającym możliwość tworzenia obiektów tej klasy do jednej instancji, dzięki czemu zapewnia globalny dostęp do stworzonego obiektu. Do drugiej grupy w tym pakiecie należą klasy ściśle związane z bazą danych, mianowicie *DataBaseAdapter* i *DataBaseHandler* oraz z obiektami, które z niej otrzymujemy i na których opiera się aplikacja, czyli *Song* oraz *Playlist*.

Tabela 3. Klasa *PlayerService*

```
public class PlayerService extends Service {

        private final IBinder mBinder = new MyBinder();

    // BROADCAST ACTIONS
    public static final String BROADCAST_ACTION = "lost connection";
    public static final String BROADCAST_ACTION_SEARCH_DEVICE = "search device";
    public static final String BROADCAST_ACTION_FOUND_DEVICE = "found device";
    public static final String BROADCAST_ACTION_SEARCH_END = "search end";
    public static final String BROADCAST_ACTION_CONNECTED = "connected";
    public static final String BROADCAST_ACTION_NOT_CONNECTED = "not connected";
    public static final String BROADCAST_ACTION_GO_TO_ALL_PLAYLISTS = "all playlists";
    public static final String BROADCAST_ACTION_GO_TO_CURRENT_PLAYLIST = "current
playlis";
    public static final String BROADCAST_ACTION_GO_TO_MUSIC_PLAYER = "music player";

    public class MyBinder extends Binder {
        PlayerService getService() {
                return PlayerService.this;
        }
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        return Service.START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
         super.onDestroy();
    }

    @Override
        public IBinder onBind(Intent arg0) {
        return mBinder;
    }

    @Override
    public void onCreate() {
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();

        if (mBtAdapter == null) {
               Toast.makeText(getApplicationContext(), BLUETOOTH_NOT_SUPPORTED,
                    Toast.LENGTH_SHORT).show();
        } else {
            if (!mBtAdapter.isEnabled()) {
               mBtAdapter.enable();
            }
        }

        // Register for broadcasts when a device is discovered
        IntentFilter filter = new IntentFilter();
        filter.addAction(BluetoothDevice.ACTION_FOUND);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        registerReceiver(mReceiver, filter);

        mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);

        sendBroadcast(new Intent().setAction(BROADCAST_ACTION_SEARCH_DEVICE));

        doDiscovery();

        if (mChatService == null){
            mChatService = new BluetoothChatService(this, mHandler);
        }

        super.onCreate();
    }

    (...)
}
```

Ostatnim z pakietów jest *com.smartwatchmusicplayer.services*, do którego należą klasy odpowiadające za połączenie Bluetooth oraz działanie aplikacji w tle. Pierwsze zadanie umożliwia *BluetoothChatService*, która zostanie opisana w dalszej części, natomiast drugie *PlayerService* (tabela 3). Klasa ta rozszerza standardową klasę dostępną w Androidzie – *Service*, która wiąże się z pojęciem już wcześniej omawianym, a mianowicie usługą. W tym wypadku będzie to usługa lokalna, gdyż jest wykorzystywana tylko na potrzeby tej konkretnej aplikacji. *PlayerService* odpowiada za komunikacje kontrolerów z *BluetoothChatService*, dzięki czemu nawet po wyjściu z aplikacji użytkownik dalej będzie informowany o ewentualnych zmianach w połączeniu Bluetooth, a w przypadku powrotu do aplikacji nie będzie musiał ponownie zestawiać tegoż połączenia. Jak widać w tabeli 3, klasa ta powiadamia kontrolery za pomocą wbudowanej funkcji *sendBroadcast()*, wysyłając adekwatne komunikaty, które są zmiennymi publicznymi.

TWORZENIE BAZY DANYCH

Tabela 4. Tabela SONG z przykładowymi danymi

	id	ARTIST	TITLE	TIME	ID_PLAYLIST
1	1	The Caesars	Jerk it out	243	4
2	2	Lana del Rey	Born to Die	198	4
3	3	Empire of the sun	Standing on the sho	221	4
4	4	Lana del Rey	Carmen	239	4
5	5	James Russo	Winged	187	2
6	6	Luis Bacalov & Rocky	Django	267	2
7	7	Ennio Morricone	The Braying Mule	206	2
8	8	Christoph Waltz & Ja	In the Case Django,	253	2
9	9	Luis Bacalov & Edda	Lo Chiamavano King	311	2
10	10	Anthony Hamilton &	Freedom	271	2
11	11	Don Johnson & Chris	Five-Thousand-Dolla	230	2
12	12	Luis Bacalov	La Corsa (2nd Versic	234	2
13	13	Don Straud	Sneaky Schultz and	211	2
14	14	Bob Marley	One Love	197	3
15	15	Coldplay	Paradise	255	3

Tabela 5. Tabela PLAYLIST z przykładowymi danymi

	id	NAME
1	1	Rap
2	2	DJANGO
3	3	Party
4	4	Car

Baza danych użyta w aplikacji składa się z dwóch tabel: SONG oraz PLAYLIST. Pierwsza z tabel przechowuje wszystkie dostępne piosenki (patrz tabela 4). Jak można zauważyć w skład tabeli SONG wchodzi pięć pól. Kolumna TITLE zawiera tytuł utworu, zaś ARTIST nazwę artysty. Pole ID_PLAYLIST jest kluczem obcym tabeli PLAYLIST i informuje, do której listy odtwarzania należy dana piosenka. Ostatnia z kolumn, czyli TIME określa czas utworu w sekundach. Druga z tabel, czyli PLAYLIST (patrz tabela 5), jak sama nazwa wskazuje zawiera dane o dostępnych *playlistach*. W jej skład wchodzą dwa pola, przy czym NAME jest nazwą konkretnej listy odtwarzania. Analogiczne do nazw powyżej opisanych tabel nazwane zostały klasy przetrzymujące obiekty z bazy danych. Czyli tymi klasami są *Song* oraz *Playlist*, które są przedstawione na odpowiednio w tabeli 6 oraz tabeli 7.

Tabela 6. Klasa *Song*

```
public class Song {

	private int id;
	private String title;
	private String artist;
	private int time;
	private int id_playlist;

	public Song() {

	}

	public Song(int _id, String _title, String _artist, int _time, int _id_playlist)
{
		setId(_id);
		setTitle(_title);
		setArtist(_artist);
		setTime(_time);
		setIdPlaylist(_id_playlist);
	}

	(..)

}
```

Tabela 7. Klasa *Playlist*

```
public class Playlist {

	private int id;
	private String name;

	public Playlist(int _id, String _name) {
		setId(_id);
		setName(_name);
	}

	(..)

}
```

Tabela 8. Klasa *DataBaseHandler*

```
	@Override
	public void onCreate(SQLiteDatabase db) {
		// TODO Auto-generated method stub
		db.execSQL("CREATE TABLE " + DATABASE_TABLE_SONG + " (" +
				KEY_ROWID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
				KEY_TITLE + " TEXT NOT NULL, " +
				KEY_ARTIST + " TEXT NOT NULL, " +
				KEY_TIME + " INTEGER NOT NULL, "+
				KEY_ID_PLAYLIST + " INTEGER NOT NULL );"
		);
		db.execSQL("CREATE TABLE " + DATABASE_TABLE_PLAYLIST + " (" +
				KEY_ROWID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
				KEY_NAME + " TEXT NOT NULL );"
		);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		// TODO Auto-generated method stub
		db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE_SONG);
		db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE_PLAYLIST);
		onCreate(db);
	}
```

Po zaprezentowaniu, jak ma wyglądać baza danych oraz jakie obiekty dzięki temu otrzymamy, zostanie opisana klasą tworzącą lokalną bazę danych, którą w omawianej aplikacji jest klasa *DataBaseHandler*, Odpowiedzialna jest ona również za aktualizację tej bazy. Metodami za to odpowiedzialnymi są *onCreate()* oraz *onUpgrade()* którą przedstawiono w tabeli 8. Obydwie należą do klasy *SQLiteOpenHelper*. Jako parametr otrzymują obiekt *SQLiteDatabase*, który reprezentuje używaną bazę danych. Metoda *onCreate()* zawiera mechanizm tworzenia tabeli i zostaje wywołana w momencie, gdy baza danych jest tworzona po raz pierwszy, natomiast *onUpgrade()* będzie użyta podczas jej aktualizacji.

Klasą odpowiedzialną za dodawanie nowych wpisów do tabel oraz wczytywanie odpowiednich danych jest klasa *DataBaseAdapter*. Metodami, należącymi do niej, a umożliwiającymi dodawanie utworów oraz *playlist* są *addSong()* oraz *addPlaylist()* (tabela 9), które jako parametr potrzebują odpowiednio obiektu klasy *Song* i *Playlist*.

Informacje, które trafiają do tej bazy, są danymi odebranymi od odtwarzacza w postaci pliku XML, który następnie jest przetwarzany. Dane wczytywane z bazy danych są za pomocą metod *getSongs()* oraz *getPlaylists()* (tabela 10). Dzięki nim otrzymujemy listę obiektów typu *Song* w przypadku pierwszej oraz typu *Playlist* dla drugiej.

Tabela 9. Metody klasy *DataBaseAdapter - addSong()* oraz *addPlaylist()*

```
public long addSong(Song song) {
          ContentValues values = new ContentValues();
          values.put(mDataBaseHandler.COLUMN_TITLE, song.getTitle());
          values.put(mDataBaseHandler.COLUMN_ARTIST, song.getArtist());
          values.put(mDataBaseHandler.COLUMN_TIME, song.getTime());
          values.put(mDataBaseHandler.COLUMN_ID_PLAYLIST, song.getId_playlist());

          return database.insert(mDataBaseHandler.DATABASE_TABLE_SONG, null,
                        values);
   }

public long addPlaylist(Playlist playlist) {
          ContentValues values = new ContentValues();
          values.put(mDataBaseHandler.COLUMN_NAME, playlist.getName());

          return database.insert(mDataBaseHandler.DATABASE_TABLE_PLAYLIST, null,
                        values);
   }
```

Tabela 10. Metody klasy *DataBaseAdapter - getSongs()* oraz *getPlaylist()*

```
 public List<Song> getSongs() {
       List<Song> songs= new ArrayList<Song>();

          Cursor cursor = database.query(mDataBaseHandler.DATABASE_TABLE_SONG,
           allSongsColumns, null, null, null, null, null);

       cursor.moveToFirst();
       while (!cursor.isAfterLast()) {
         Song song = cursorToSong(cursor);
         songs.add(song);
         cursor.moveToNext();
       }
       // make sure to close the cursor
       cursor.close();
       return songs;
     }

 public List<Playlist> getPlaylists() {
       List<Playlist> playlists= new ArrayList<Playlist>();

          Cursor cursor = database.query(mDataBaseHandler.DATABASE_TABLE_PLAYLIST,
           allPlaylistsColumns, null, null, null, null, null);

       cursor.moveToFirst();
       while (!cursor.isAfterLast()) {
         Playlist playlist = cursorToPlaylist(cursor);
         playlists.add(playlist);
         cursor.moveToNext();
       }
       // make sure to close the cursor
       cursor.close();
       return playlists;
     }
```

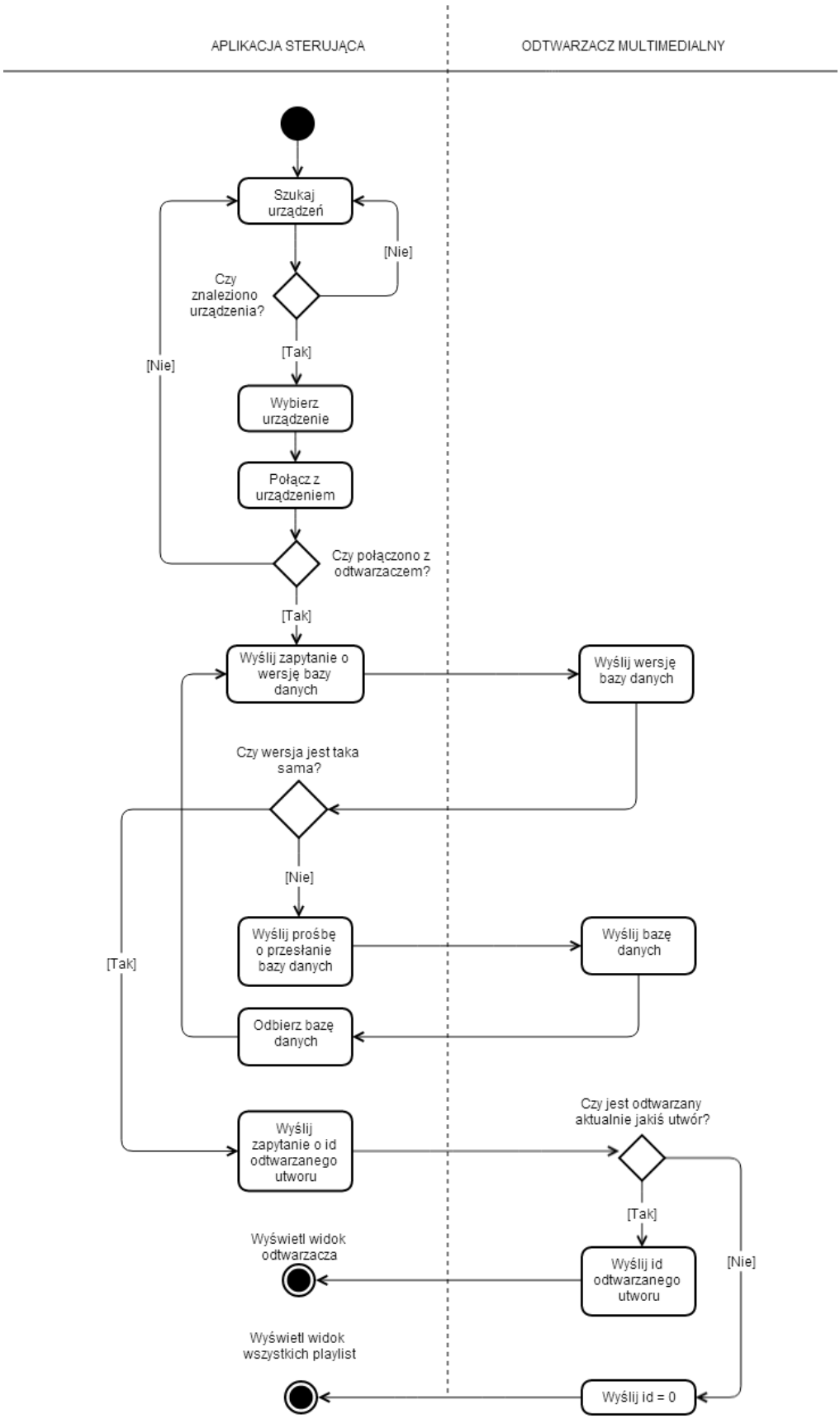
KOMUNIKACJA Z ODTWARZACZEM MULTIMEDIALNYM

Po przedstawieniu jak tworzona jest baza danych, nadszedł czas, by wyjaśnić jak wygląda komunikacja z odtwarzaczem multimedialnym, która jest możliwa za pomocą technologii Bluetooth. Początkowo aplikacje miały wykorzystywać Bluetooth w wersji 4.0. Charakteryzuje się bardzo niskim zużyciem energii w stosunku do poprzednich wersji, wobec czego inaczej nazywana jest *Low Energy*. Byłoby to idealne rozwiązanie dla zegarków elektronicznych, gdyż wtedy program nie wpływałby tak znacząco na zużycie baterii. Niestety obecna wersja Androida nie pozwala w pełnym stopniu na wprowadzenie technologii energooszczędnej, ponieważ obecnie można jedynie odbierać komunikaty, natomiast nie można nic nadawać. Dlatego też została zastosowana niższa wersja, a z Bluetooth 4.0 będzie można skorzystać w przyszłości.

Schemat pokazujący jak jest nawiązywane połączenie pomiędzy aplikacjami przedstawia rysunek 7. Wynika z tego, że na początku program na zegarku elektronicznym szuka widocznych urządzeń, a gdy już takowe znajdzie, wyświetla je użytkownikowi, który może je wybrać. Następnie po prawidłowym połączeniu z odtwarzaczem następuje wymiana komunikatów dotyczących wersji bazy danych, a w przypadku jej niezgodności przesyłana jest cała baza w postaci danych XML. Po czym ponownie jest sprawdzana wersja bazy danych, w celu weryfikacji, czy nie miały miejsca żadne zmiany. Następnie aplikacja na zegarku odpytuje drugi program o obecnie odtwarzany utwór. Jeśli jest takowy, to zostanie przesłane jego *id*, a w przeciwnym razie odpowiedzią będzie 0, gdyż taka wartość nie może wystąpić w bazie danych. Adekwatnie do otrzymanego *id* użytkownik zobaczy albo ekran odtwarzacza (*id* różne od zera) albo wszystkie *playlisty* (id równe zero). Następnie aplikacje wymieniają między sobą komunikaty, których lista została zaprezentowana poniżej:

- **MESSAGE_TYPE_PLAY** – wznowienie odtwarzania
- **MESSAGE_TYPE_PAUSE** – zatrzymanie odtwarzania
- **MESSAGE_TYPE_PREVIOUS** – zmiana utworu na poprzedni
- **MESSAGE_TYPE_NEXT** – zmiana utworu na następny
- **MESSAGE_TYPE_VOLUME_UP** – podgłośnienie
- **MESSAGE_TYPE_VOLUME_DOWN** – ściszenie

- **MESSAGE_TYPE_SEND_SONG_ID** – przesłanie id utworu wybranego z *playlisty*
- **MESSAGE_TYPE_GET_SONG_ID –** zapytanie o id aktualnie odtwarzanego utworu
- **MESSAGE_TYPE_GET_DB_VERSION –** zapytanie o aktualną wersję bazy danych
- **MESSAGE_TYPE_SEND_DB_VERSION –** przesłanie wersji bazy danych
- **MESSAGE_TYPE_GET_DB –** prośba o przesłanie bazy danych
- **MESSAGE_TYPE_SEND_DB –** przesłanie bazy danych



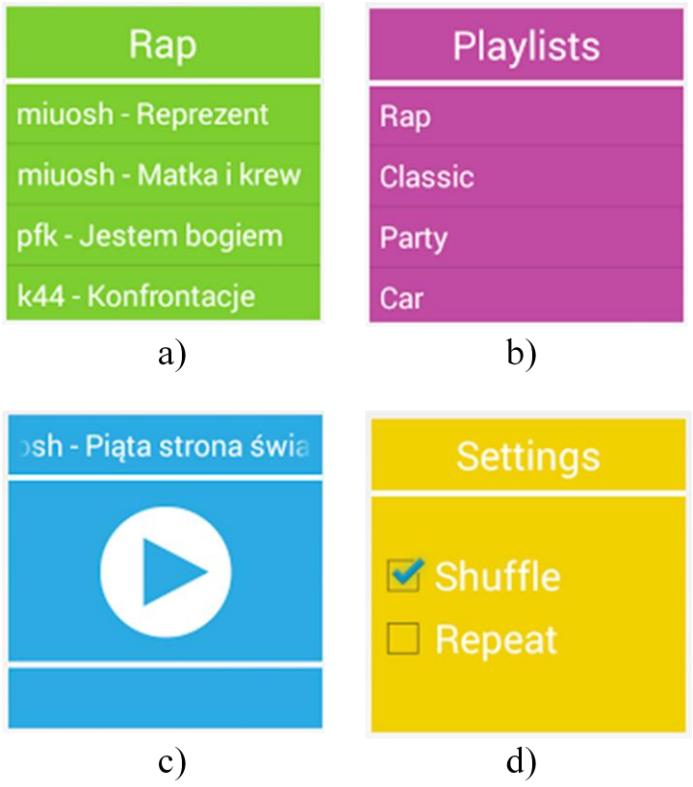
Rysunek 7. Diagram aktywności (zestawienie połączenia Bluetooth)

INTERFEJS GRAFICZNY

Jak już wcześniej było wspomniane aplikacja składa się z pięciu widoków. Do kontrolerów odpowiedzialnych za nie należą:

- ***MusicPlayerController*** (rysunek 8c) – pozwala na wyświetlanie tytułu odtwarzanego utworu a użytkownikowi umożliwia jego zmianę, zatrzymanie, wznowienie, bądź kontrolę głośności.
- ***CurrentPlaylistController*** (rysunek 8a) – pokazuje aktualną *playlistę*, czyli dostępne w niej utwory, w formie listy, dzięki wykorzystaniu standardowej klasy znajdującej się w Androidzie – *ListView*. Pozwala na zmianę utworu w obrębie tej *playlisty*.
- ***AllPlaylistsController*** (rysunek 8b) – wyświetla wszystkie dostępne listy odtwarzania, w taki sam sposób jak poprzednia klasa, wobec czego tutaj można wybrać interesującą nas *playlistę*.
- ***SettingsController*** (rysunek 8d) – umożliwia użytkownikowi wybranie opcji zapętlenia utworu (*Repeat*) oraz odtwarzania losowego (*Shuffle*).
- ***DeviceListController*** – pojawia się wtedy i tylko wtedy, gdy urządzenie wyszukuje odtwarzacz multimedialny za pomocą Bluetooth i odpowiada za wyświetlenie znalezionych urządzeń.

Ponadto z powyższymi klasami skorelowana jest klasa *OnSwipeTouchListener*, dzięki której użytkownik może poruszać się pomiędzy widokami za pomocą gestów tzw. *swipe'ów*. *Swipe* oznacza dłuższe przeciągnięcie przez użytkownika palcem po wyświetlaczu w odpowiednim kierunku. W związku z tym nawigacja w aplikacji następuje za pomocą takich diagonalnych gestów i tak przesuwając w lewy górny róg włączymy widok odtwarzacza, lewy dolny – obecną *playlistę*, prawy dolny – wszystkie *playlisty* oraz prawy górny – ustawienia. Ponadto gesty tego typu użyte są również w pierwszym z tych widoków do kontroli odtwarzacza. *Swipe* w górę, bądź dół (przesunięcie pionowo palcem) odpowiada za zmianę głośności, odpowiednio głośniej i ciszej, natomiast *swipe* w lewo i prawo (przesunięcie poziomo palcem) za zmianę utworów – poprzedni, następny.



Rysunek 8. Widoki aplikacji sterującej odtwarzaczem

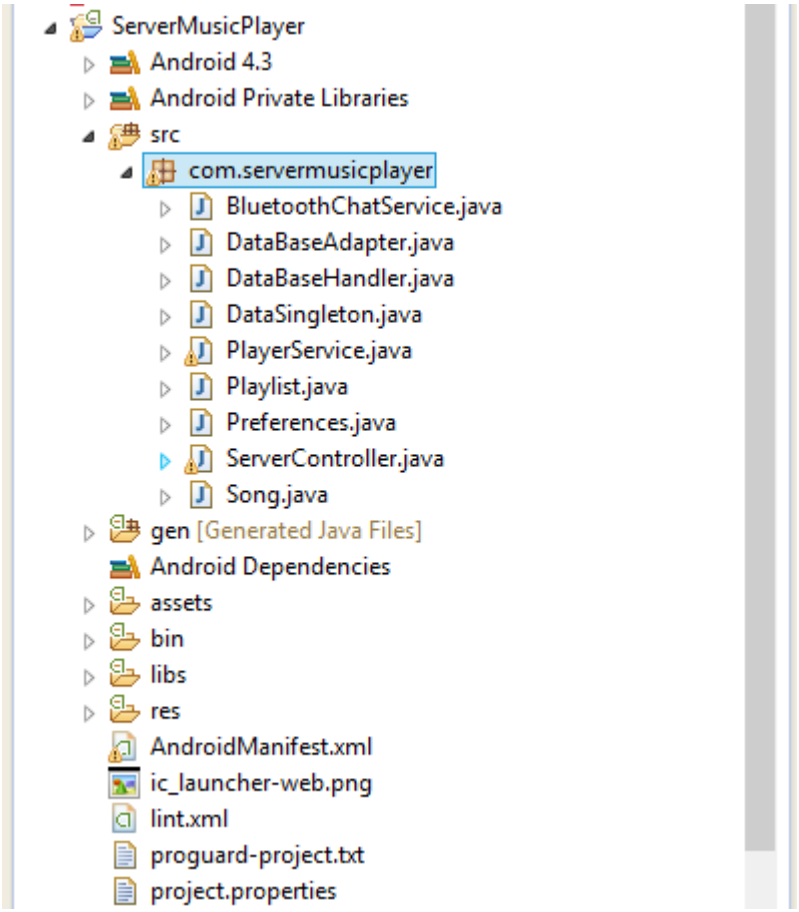
2.2. Odtwarzacz multimedialny

W niniejszym podrozdziale zostanie szczegółowo opisana struktura odtwarzacza multimedialnego, sposób w jaki są strumieniowane dane z serwera oraz jak tworzona jest baza danych.

STRUKTURA APLIKACJI

Program został zbudowany w oparciu o osiem klas, które wchodzą w skład pakietu *com.servermusicplayer* (rysunek 9). Aż siedem z nich jest nazwane tak samo, jak w poprzedniej aplikacji, czyli *DataSingleton*, *DataBaseHandler*, *DataBaseAdapter*, *PlayerService*, *BluetoothChatService*, *Song* oraz *Playlist*, co świadczy o tym, iż są odpowiedzialne za podobne, bądź takie same funkcje. Dodatkowo pojawia się klasa

ServerController, której zadaniem jest uruchomienie usługi lokalnej, aby aplikacja działała w tle.



Rysunek 9. Struktura programu odtwarzacza multimedialnego

TWORZENIE BAZY

Baza danych wykorzystana w tej aplikacji składa się z dokładnie takich samych tabel, jak w poprzednim programie, czyli SONG (patrz tabela 5) oraz PLAYLIST, które zawierają identyczne pola. Jedyną różnicą jest dodatkowa kolumna LINK występująca w tabeli SONG. Zawiera ona informacje pod jakim adresem URL znajduje się dany utwór. Przykładowe informacje zawarte są w tabeli 11.

Baza danych tworzona jest w analogiczny sposób i za pomocą tych samych metod, które również występują w klasie *DataBaseHandler* (patrz tabela 8). Również metody odpowiedzialne za dodawanie wpisów do tabel oraz uzyskiwanie z nich danych są prawie identyczne (patrz tabela 9 i tabela 10) i zawarte w klasie *DatabseAdapter*.

Tabela 11. Tabela SONG z przykładowymi danymi

	id	LINK	ARTIST	TITLE	TIME	ID_PLAYLIST
1	1	http://pluton.kt.agh.edu	The Caesars	Jerk it out	243	4
2	2	http://pluton.kt.agh.edu	Lana del Rey	Born to Die	198	4
3	3	http://pluton.kt.agh.edu	Empire of the sun	Standing on the sho	221	4
4	4	http://pluton.kt.agh.edu	Lana del Rey	Carmen	239	4
5	5	http://pluton.kt.agh.edu	James Russo	Winged	187	2
6	6	http://pluton.kt.agh.edu	Luis Bacalov & Rocky	Django	267	2
7	7	http://pluton.kt.agh.edu	Ennio Morricone	The Braying Mule	206	2
8	8	http://pluton.kt.agh.edu	Christoph Waltz & Ja	In the Case Django,	253	2
9	9	http://pluton.kt.agh.edu	Luis Bacalov & Edda	Lo Chiamavano King	311	2
10	10	http://pluton.kt.agh.edu	Anthony Hamilton &	Freedom	271	2
11	11	http://pluton.kt.agh.edu	Don Johnson & Chris	Five-Thousand-Dolla	230	2
12	12	http://pluton.kt.agh.edu	Luis Bacalov	La Corsa (2nd Versic	234	2
13	13	http://pluton.kt.agh.edu	Don Straud	Sneaky Schultz and	211	2
14	14	http://pluton.kt.agh.edu	Bob Marley	One Love	197	3
15	15	http://pluton.kt.agh.edu	Coldplay	Paradise	255	3

Dane, które trafiają do odpowiednich tabel to utwory muzyczne znajdują się na prywatnym zewnętrznym serwerze, z którego aplikacja pobiera plik XML w oparciu, którego budowana jest opisana powyżej baza danych. Ów plik, który jest przedstawiony z przykładowymi danymi w tabeli 12, składa się z elementu głównego *playlists*, który zawiera elementy podrzędne *playlist* wchodzące w skład tabeli PLAYLIST. Zawierają one atrybuty *name*, czyli nazwy tych list odtwarzania. Każdy element *playlist* składa się z elementów podrzędnych *song*, czyli piosenki będącej wpisem w tabeli SONG. Elementami podrzędnymi w tym wypadku są: *title*, *artist*, *time* oraz *link*, które nazywają się dokładnie tak samo jak kolumny tej tabeli. Po pobraniu ten plik jest przetwarzany, aby dane mogły trafić do odpowiednich tabel w lokalnej bazie danych. Jednakże rozwiązanie korzystające właśnie z pliku XML nie jest najlepsze. Wynika to z faktu, iż byłoby to uciążliwe dla użytkownika końcowego, który musiałby zmieniać jego zawartość wraz z każdą zmianą utworów na serwerze. Odpowiednim sposobem było korzystanie z funkcji umożliwiających odpytanie serwera, co do jego zawartości, np. za pomocą metody GET.

Tabela 12. Konfiguracyjny plik XML z przykładowymi danymi

```
<?xml version="1.0" encoding="UTF-8"?>
<playlists>
    <playlist name = "Car">
        <song>
            <title>Carmen</title>
            <artist>Lana del Rey</artist>
            <time>267</time>
            <link>http://pluton.kt.agh.edu.pl/09%20Carmen.mp3</link>
        </song>
        <song>
            <title>Born to die</title>
            <artist>Lana del Rey</artist>
            <time>283</time>
            <link>http://pluton.kt.agh.edu.pl/01%20Born%20to%20Die.mp3</link>
        </song>
    </playlist>
</playlists>
```

STRUMIENIOWANIE ZEWNĘTRZNYCH DANYCH Z SERWERA

Utwory muzyczne odtwarzane w tej aplikacji, jak już było wcześniej wspomniane, znajdują się na dedykowanym serwerze. Dzięki temu, że w bazie danych mamy zapisany adres każdego z nich możliwe jest ich strumieniowanie, co wykorzystano w programie. Odbywa się to w klasie *PlayerService* za pomocą obiektu typu *MediaPlayer* (standardowej klasy dostępnej w Androidzie). Metodami wykorzystanymi w tym celu są *setDataSource(), prepare()* oraz *start()*. Parametrem pierwszej z nich jest właśnie adres piosenki. Następna z metod odpowiada za przygotowanie instancji tej klasy do odtwarzania, które jest rozpoczęte przez ostatnią z nich. W omawianej klasie pojawiają się również metody odpowiedzialne za zmianę piosenki na następna oraz poprzednią – *changeSong()* (tabela 13) której parametrem jest odpowiedni obiekt z listy piosenek *Songs*. W tej metodzie najpierw następuje zatrzymanie odtwarzacza – funkcja *stop()*, który następnie jest zresetowany – *reset(),* po czym ponownie wykorzystane są *setDataSource(), prepare()* oraz *start().*

Kompletny kod źródłowy obydwu aplikacji wchodzących w skład opisywanego systemu został umieszony w Dodatku A niniejszej pracy.

Tabela 13. Metoda *changeSong()*

```
public void changeSong(){

            mp.stop();
            sendMessageToDevice(MESSAGE_TYPE_PAUSE);

            if (mp != null) {
          try {
            boolean isPlaying = mp.isPlaying();
                    mp.reset();
                            mp.setDataSource(this, Uri.parse(title[currentSong]));
                    mp.prepare();
                    if(isPlaying){
                            mp.start();
                    }
                    } catch (IllegalArgumentException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    } catch (SecurityException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    } catch (IllegalStateException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }
    }

    sendMessageToDevice(MESSAGE_TYPE_SEND_SONG_ID);
}
```

3. Testy aplikacji

Niniejsze aplikacje wchodzące w skład opisywanego systemu testowane były na *smartfonie* Samsung Galaxy SIII (program sterujący) oraz na przystawce Measy U2C (odtwarzacz multimedialny). Niestety nie skorzystano z *smartwatcha* ze względu na jego niedostępność, jednakże użyte urządzenie odpowiednio go zasymulowało. Aplikacje były testowane pod kątem szybkości, czytelności oraz czułości na odpowiednie gesty.

Mając na uwadze pierwszy z tych aspektów, czyli szybkość, można wyodrębnić dwie kategorie, a mianowicie komunikację pomiędzy urządzeniami oraz strumieniowanie. Aplikacje wykorzystają do połączenia między sobą Bluetooth, dlatego też, gdy znajdują się w odpowiedniej odległości, to komunikacja pomiędzy nimi działa bez zarzutu, a przede wszystkim szybko – około kilku milisekund. Również biorąc pod uwagę streaming podczas testowania nie pojawiły się żadne opóźnienia, wobec czego można stwierdzić, że dopóki jest połączenie pomiędzy serwerem a urządzeniem, to system działa bez zarzutów.

Czułość na gesty, była testowana tylko na aplikacji przeznaczonej na *smartwatche*. W tym celu dziesięć osób przez trzy minuty miało możliwość korzystania z tego programu. Następnie w skali od 1 do 5 (najlepiej) ocenili użyteczność przycisków (wznów oraz zatrzymaj odtwarzanie, włącz/wyłącz odtwarzanie losowe i zapętlenie), czułość na dostępne gesty (zmiana głośności oraz utworu na poprzedni i następny, włączenie innych widoków: Odtwarzacz, Aktualna playlista, Wszystkie playlisty i Ustawienia) oraz czytelność napisów, co zebrano i przedstawiono w tabeli 14. Z przeprowadzonych testów wynika, że użyteczność przycisków jest na odpowiednim poziomie, gdyż użytkownicy ocenili ją średnio na 5 (odtwarzaj oraz zastopuj) oraz 4.5 (odtwarzanie losowe i zapętlenie). Niestety gorzej już prezentują się oceny dla możliwości sterowania odtwarzaczem za pomocą gestów. Zmiana utworów została oceniona na zadowalającym poziomie tj. średnio 4.2 zarówno dla następnego, jak i poprzedniego utworu. Jednakże słabsze noty uzyskała kontrola głośności – 3.9 dla ściszania oraz 3.3 dla pogłaśniania. Z pewnością na takie oceny ma wpływ fakt, iż podczas wykonywania gestów *swipe* w górę lub dół może zostać przypadkowo kliknięty przycisk odtwarzaj/pauza. Oceny użytkowników dla zmiany widoków są zależne od

tego, w którą stronę mamy wykonać gest, dlatego najwyższą średnią ma Odtwarzacz – 4.8 (lewy górny róg), natomiast najniższą Ustawienia – 4.2 (prawy górny róg). Czytelność napisów w aplikacji została oceniona wysoko – średnia 4.9, wobec czego można uznać, iż w aplikacji została wykorzystana odpowiednia wielkość czcionki. Jednakże aplikacja testowana była na *smartfonie* o przekątnej wyświetlacza 4.8”, co mogło znacząco poprawić wynik w tej kwestii, ponieważ dostępne zegarki mają ekrany o wielkości 1.6”.

Tabela 14. Oceny użytkowników

	Odtwarzaj	Stop	Odtwarzanie losowe	Zapętlenie	Głośniej	Ciszej	Następny	Poprzedni	Odtwarzacz	Aktualna playlista	Wszystkie playlisty	Ustawienia	Czytelność
Użytkownik 1	5	5	5	5	4	4	4	4	5	5	4	4	5
Użytkownik 2	5	5	5	5	4	5	5	5	5	5	4	4	5
Użytkownik 3	5	5	5	5	4	4	5	5	5	5	5	5	5
Użytkownik 4	5	5	4	4	3	4	4	4	5	5	4	4	5
Użytkownik 5	5	5	4	4	2	3	4	4	5	4	4	5	5
Użytkownik 6	5	5	4	4	3	3	4	4	4	5	3	4	4
Użytkownik 7	5	5	4	4	2	3	3	3	4	4	5	4	5
Użytkownik 8	5	5	5	5	4	5	5	5	5	5	5	4	5
Użytkownik 9	5	5	5	5	4	4	4	4	5	5	4	4	5
Użytkownik 10	5	5	4	4	3	4	4	4	5	4	5	4	5
Średnia	**5**	**5**	**4.5**	**4.5**	**3.3**	**3.9**	**4.2**	**4.2**	**4.8**	**4.7**	**4.3**	**4.2**	**4.9**

Podsumowanie

Celem niniejszej pracy inżynierskiej było zaprojektowanie oraz stworzenie systemu, który umożliwiałby użytkownikowi w jak najprostszy sposób zarządzania odtwarzaczem multimedialnym. W jego skład wchodzą dwie aplikacje, z których pierwsza odpowiada za sterowanie i jest zaimplementowana na *smartwatche* z Androidem. Oferuje ona następujące funkcjonalności: wybór utworu, stop, odtwarzaj, następny, poprzedni, ciszej, głośniej. Ponadto cechuje się bardzo minimalistycznym interfejsem, przez co jest intuicyjna oraz prosta w obsłudze dla nawet najmniej zaawansowanego użytkownika. Druga aplikacja jest odpowiedzialna za odtwarzanie utworów wybranych przez użytkownika za pomocą strumieniowania zewnętrznych danych z serwera.

System testowany był przez grupę dziesięciu użytkowników, którzy oceniali aplikację na zegarki elektroniczne pod kątem użyteczności przycisków, czułości na dostępne gesty oraz czytelności. Zarówno pierwszy, jak i ostatni z tych aspektów otrzymały ocenę prawie bardzo dobrą, natomiast drugi dobrą.

Zrealizowany projekt inżynierski podyktowany był rosnącą popularnością urządzeń typu wearable technology, a w tym konkretnym wypadku *smartwatchy* i ma za zadanie przedstawienie pracy w środowisku Eclipse przy programowaniu aplikacji na system Android. Praca ta nie wyczerpuje w pełni tematu. System ma perspektywy rozwoju w wielu kierunkach, z których najważniejsze to interfejs graficzny drugiej aplikacji umożliwiający odtwarzania filmów, wykorzystanie popularnych serwerów takich jak Dropbox, czy Google Drive oraz wprowadzenie do komunikacji Bluetooth 4.0.

Literatura

[1] S. Hashimi, S. Komatineni, D. MacLean, *Android 2. Tworzenie aplikacji*, wydawnictwo Helion, 2010.

[2] Android Developer. [Online]. Dostępny w Internecie: http://www.developer.android.com

[3] Dokumentacja Bluetooth. [Online]. Dostępny w Internecie: https://developer.bluetooth.org

[4] *Smartwatch forecast 2013-2020.* [Pdf]

[5] A. Żółciak, *Smartwatch zmieni rynek mobilny?* [Online]. 2013-03-24 [dostęp: 13. grudnia 2013]. Dostępny w Internecie: http://www.gsmmaniak.pl/192654/smartwatch-zmieni-rynek-mobilny/

[6] S. Conder, L. Darcey, *Android. Programowanie aplikacji na urządzenia przenośne. Wydanie II*, wydawnictwo Helion.

[7] Produkty firmy Measy. [Online]. Dostępny w Internecie: http://www.measypolska.pl/p/blog-page.html

[8] Produkty firmy Cabletech. [Online]. Dostępny w Internecie: http://www.cabletech.pl/pl/1292-multimedia-do-telewizji-

[9] Specyfikacja Omate Truesmart. [Online]. Dostępny w Internecie: http://www.kickstarter.com/projects/omate/omate-truesmart-water-resistant-standalone-smartwa

[10] Specyfikacja i'm Watch . [Online]. Dostępny w Internecie: http://www.imsmart.com/pl/i-m-watch/specifications

[11] Specyfikacja Neptune Pine, Samsung Galaxy Gear, Sony Smartwatch 2. [Online]. Dostępny w Internecie: http://www.kickstarter.com/projects/neptune/neptune-pine-smartwatch-reinvented

Dodatek A. Spis zawartości dołączonej płyty CD

Praca inżynierska – niniejszy dokument

Kod źródłowy obydwu aplikacji