



POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI
KIERUNEK INFORMATYKA

Projekt inżynierski

System akwizycji i udostępniania danych z czujników EEG

Autor: Mateusz Boś

Kierujący pracą: dr inż. Krzysztof Tokarz

Konsultant: mgr inż. Marcin Wierzchanowski

Gliwice, styczeń 2014

Spis treści

1	Wstęp	1
2	Analiza tematu	2
2.1.1	Pojęcia ogólne	2
2.2	EEG, fale mózgowe	3
2.2.1	Alfa	3
2.2.2	Beta	3
2.2.3	Delta	3
2.2.4	Gamma	3
2.2.5	Theta	3
2.3	EMG, napięcie mięśniowe	5
2.4	Wymagania stawiane aplikacji	5
2.5	Wykorzystywane narzędzia	5
3	Organizacja wewnętrzna	6
3.1	Aplikacja mobilna	6
3.1.1	Ogólny opis	6
3.1.2	Struktura i działanie procesu akwizycji	7
3.1.3	Struktura i działanie procesu danych	9
3.1.4	Komunikacja międzyprocesowa	12
3.1.5	Organizacja bazy danych	15
3.2	Aplikacja serwerowa	16
3.2.1	Ogólny opis	16
3.2.2	Udostępniane punkty końcowe	16
3.2.2.1	Logowanie	16
3.2.2.2	Synchronizacja różnicowa	16
3.2.2.3	Zapisywanie danych	17
3.2.2.4	Weryfikacja serwisu	18
3.2.2.5	Podgląd danych	18
3.2.3	Organizacja bazy danych	19
4	Implementacja	20
4.1	Aplikacja mobilna	20
4.1.1	Użyte biblioteki	20
4.1.2	Szczegóły implementacji	21
4.2	Aplikacja serwerowa	26
4.2.1	Użyte biblioteki	26
4.2.2	Szczegóły implementacji	26
5	Testowanie i uruchamianie	29

5.1	Model wzorcowy	29
5.2	Testowanie aplikacji mobilnej	30
5.3	Testowanie aplikacji serwerowej	30
5.4	Uruchamianie aplikacji pracujących wspólnie	32
6	Specyfikacja zewnętrzna	33
6.1	Aplikacja mobilna	33
6.1.1	Instalacja	33
6.1.1.1	Nieznane źródła	33
6.1.1.2	Z linii poleceń	33
6.1.1.3	Poprzez eksplorator plików	34
6.1.2	Użycie	36
6.2	Aplikacja serwerowa	37
6.2.1	Instalacja	37
6.2.1.1	MySQL	37
6.2.1.2	Glassfish	38
6.2.2	Użycie	38
7	Uwagi końcowe	41
8	Bibliografia	43
9	Spis rysunków	44
10	Spis fragmentów kodu	45

1 Wstęp

Ciągle postępująca miniaturyzacja elementów elektronicznych, której towarzyszy jednocześnie zmniejszenie poboru prądu i wzrost dostępnej mocy obliczeniowej, odkrywa nowe możliwości we wręcz nieskończonej ilości dziedzin. Dzięki postępującej miniaturyzacji systemy wcześniej zajmujące całe pomieszczenia, teraz osadzone są na jednym kawałku krzemu, którego rozmiar oscyluje w granicach kilkunastu milimetrów są powszechnie dostępne.

Jedną z dziedzin której rozwój uwarunkowany jest dostępnością kompleksowych układów zamkniętych w jednym opakowaniu SiP (z ang. *System in a Package*) jest mobilne EEG. Mobilne EEG może być wykorzystywane na wiele sposobów, zaczynając od rozrywki czy wspomaganiu nauczania a kończąc na zastosowaniach medycznych na przykład wspomaganiu diagnozowania chorób takich jak ADHD czy choroba Parkinsona lub rehabilitacji.

W diagnostyce chorób ważne są dane którymi dysponuje lekarz specjalista, większa ilość danych (wysokiej jakości) z dłuższego okresu czasu wspomaga specjalistę w dokonywaniu trafnej diagnozy. Wykorzystanie mobilnego EEG jest tutaj naturalnym sposobem pozwalającym nieprzerwanie monitorować pacjenta w dowolnym czasie i miejscu przez dwadzieścia cztery godziny na dobę. Daje to nieporównywalnie większą ilość danych od tradycyjnego badania EEG wymagającego obecności w placówce, pozwala dodatkowo uchwycić czasem trudne do uchwycenia anomalie. Zestaw firmy Neurosky nie wymaga nieprzyjemnej aplikacji specjalnego żelu wspomagającego przewodzenie bodźców elektrycznych (tak zwany „suchy sensor”), co również pozytywnie wpływa na komfort osoby badanej. Akwizycja danych wykonywana w taki sposób nie powinna wprowadzać praktycznie żadnych niedogodności do życia codziennego, włączając w to aplikację wcześniej wspomnianego żelu. Jednocześnie wymagania mobilnego EEG sprowadzają się tylko do posiadania smartfona z obsługą technologii Bluetooth.

Celem niniejszej pracy jest wykonanie aplikacji mobilnej pozwalającej na gromadzenie danych z czujników firmy Neurosky model MindWave Mobile oraz przekazywanie ich do aplikacji serwerowej pozwalającej przeglądać zgromadzone dane. Dokładny opis funkcjonalności wymaganych od obu części pracy przedstawiony jest w rozdziale 2.4.

2 Analiza tematu

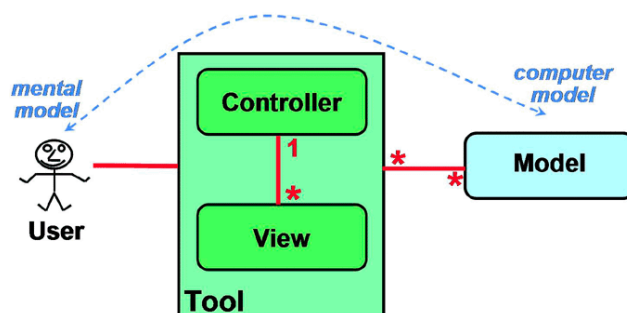
W celu pełnego zrozumienia omawianych zagadnień, w tym rozdziale przytoczono definicje używanych w pracy pojęć i narzędzi.

2.1.1 Pojęcia ogólne

Poniżej przytoczono definicję pochodzącą ze strony¹ projektu JSON (JavaScript Object Notation) jest prostym formatem wymiany danych. Zapis i odczyt danych w tym formacie jest łatwy do zrozumienia przez ludzi. Jednocześnie, z łatwością odczytują go i generują komputery. Jego definicja opiera się o podzbiór języka programowania *JavaScript, Standard ECMA-262 3rd Edition - December 1999*. JSON jest formatem tekstowym, całkowicie niezależnym od języków programowania, ale używa konwencji, które są znane programistom korzystającym z języków z rodziny C, w tym C++, C#, Java, JavaScript, Perl, Python i wielu innych. Właściwości te czynią JSON idealnym językiem wymiany danych.” Format ten używany jest do komunikacji między aplikacją na system Android a aplikacją serwerową napisaną w Javie.

W pracy do implementacji aplikacji serwerowej użyto wzorca architektury oprogramowania REST. Rozwinięciem akronimu jest *Representational State Transfer*, a pełny opis tego wzorca projektowego zawarty został w pracy [1].

Aplikację kliencką wykonano z użyciem rodzaju wzorca MVC (*Model View Controller*), który wprowadza dodatkową warstwę abstrakcji, pozwalając na ponowne użycie tworzonego kodu.



Rysunek 2-1 Koncepcja wzorca model widok kontroler, źródło²

¹ Definicja podana za <http://www.json.org/json-pl.html>, [dostęp 25 grudnia 2013].

² Strona domowa Trygve H. M. Reenskaug <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> [dostęp 25 grudnia 2013]

2.2 EEG, fale mózgowe

Głównym tematem pracy jest EEG (*Elektroencefalografia*), czyli nieinwazyjna metoda zapisu aktywności elektrycznej mózgu odkryta przez niemieckiego lekarza Hansa Bergera. W Pracy używany jest czujnik Neurosky MindWave Mobile, który powinien być umieszczany w punkcie FP1 (według międzynarodowego systemu 10-20, patrz rysunek 2-3). Ogólnie znanych jest więcej rodzajów, jednak ograniczenia sprzętowe, jakie są narzucone na Autora, pozwalają obserwować najczęściej występujące, a wymienione poniżej pięć klas.

2.2.1 Alfa

Fale, których częstotliwość zawiera się w przedziale 7,5 do 12 Hz. Powiązane ze stanami relaksacji oraz nieuwikłania w wymagające czynności. Nasilają się po zamknięciu oczu, najlepiej obserwowalne w obszarze potylicznym (punkty O1 i O2).

2.2.2 Beta

Fale z zakresu 12 do 30 Hz, często wyróżnia się fale β_1 oraz β_2 w celu uzyskania bardziej szczegółowych zakresów. Fale charakteryzują się mniejszym niż fale Alfa okresem. Ogólnie związane z skupieniem, koncentracją. Najlepiej obserwowalne w obszarach centralnym oraz przednim. Nasilają się przy wykonywaniu czynności takich jak rozwiązywanie równań matematycznych, łamigłówek czy wstrzymywanie ruchów.

2.2.3 Delta

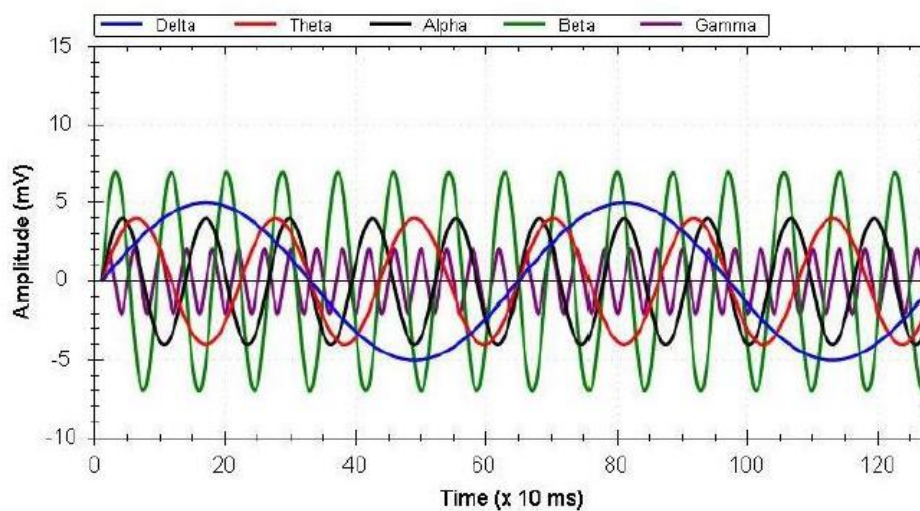
Zakres tych fal sięga od 0,5 do 3,5 Hz. Są najwolniejszymi z fal, u normalnie rozwiniętych, dorosłych osób występują jedynie podczas snu. Obecność tych fal u przytomnych osób wskazuje defekt mózgu.

2.2.4 Gamma

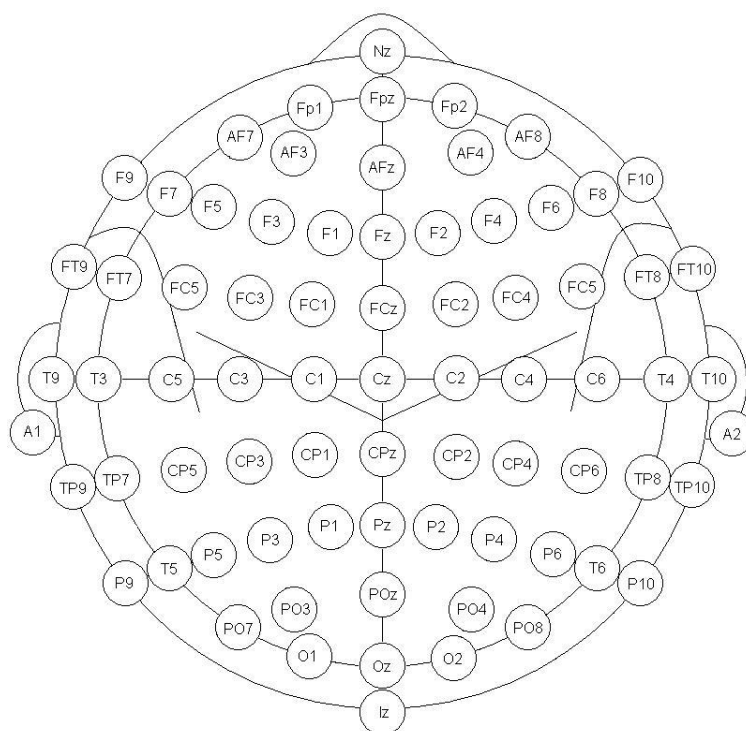
Fale z zakresu częstotliwości większego niż 31 Hz. Razem z falami beta powiązane są ze skupieniem, mechanizmami poznawania i rozpoznawania.

2.2.5 Theta

Fale, których częstotliwość zawiera się w przedziale 3,5 do 7,5 Hz. Wiązane z niewydajnością, snami na jawie. Wyzwalane przez stres emocjonalny zwłaszcza frustrację i rozczarowanie.



Rysunek 2-2 Zestawienie głównych klas fal mózgowych, źródło [2], strona 11



Rysunek 2-3 Rozmieszczenie elektrod w systemie 10-20, źródło³

³ Wiki wydziału medycznego Uniwersytetu Michigan
<https://wiki.umms.med.umich.edu/download/attachments/90734989/EEG+10-20+system+map.JPG>

2.3 EMG, napięcie mięśniowe

W pracy zbierane są informacje o mrugnięciach oczyma. Umożliwione jest to poprzez technikę znaną jako EMG, czyli Elektromiografię, diagnostykę czynności elektrycznej mięśni i nerwów obwodowych.

2.4 Wymagania stawiane aplikacji

Założenia funkcjonalne stawiane aplikacjom wyglądają następująco:

- ☐ obsługa kont użytkownika (logowanie),
- ☐ różnicowa synchronizacja danych.

Założenia niefunkcjonalne:

- ☐ zgodność z systemem Android $\geq 4.0.4$,
- ☐ implementacja GUI ,
- ☐ akwizycja danych powinna zachodzić w tle,
- ☐ możliwość wyboru kiedy powinna nastąpić synchronizacja danych.

Obrano architekturę systemu jako klient-serwer. Została ona zobrazowana na Rysunek 2-4. Architekturę tę wybrano przez wzgląd na prostotę i naturalne dopasowanie do rozpatrywanego problemu.



Rysunek 2-4 Architektura systemu, źródło własne.

2.5 Wykorzystywane narzędzia

Część aplikacyjna pracy składa się z dwóch części: aplikacji mobilnej i aplikacji serwerowej. Do napisania aplikacji serwerowej użyto środowiska Eclipse w wersji Kepler

Service Version 1, którą pobrano ze strony projektu⁴ oraz serwera Glassfish w wersji 4.0, którą również pobrano ze strony projektu⁵. Część mobilną napisano z użyciem środowiska dostarczanego przez Google i dedykowanego do pisania aplikacji na system Android – Android Studio, który można pobrać z Internetu⁶. Zdecydowano się użyć systemu Gradle, ponieważ jest on zalecany przez twórców systemu i dodatkowo jest on nowym, elastycznym systemem pozwalającym dowolnie modyfikować proces budowania. Używano go poprzez będącą w ciągłym rozwoju wtyczkę (z ang. *plugin*) do budowania aplikacji Android⁷.

3 Organizacja wewnętrzna

Ten rozdział podzielono na dwie części: pierwszą opisującą strukturę aplikacji mobilnej oraz drugą przedstawiającą budowę aplikacji serwerowej.

3.1 Aplikacja mobilna

3.1.1 Ogólny opis

W celu spełnienia założenia mówiącego o akwizycji danych w tle zdecydowano się podzielić aplikację na trzy osobne procesy, ogólny widok architektury aplikacji można zobaczyć na rysunku 3-1. Procesy zostaną opisane w takiej kolejności, w jakiej są tworzone w aplikacji. Pierwszy proces, jaki jest tworzony, to główny proces aplikacji. Nazwano go procesem UI. Jest on odpowiedzialny za obsługę interfejsu użytkownika, na którą składa się odbieranie sygnałów od użytkownika oraz wyświetlanie komunikatów przeznaczonych dla niego. Proces ten jest tworzony przy otwieraniu aplikacji i dodatkowo powołuje on do życia dwa pozostałe procesy. Pierwszy z nich, proces danych, jeśli jeszcze nie istnieje, jest powoływany do życia w momencie prezentacji ekranu logowania za pośrednictwem obiektu klasy `AuthorizationServiceClient`. Ostatnim uruchamianym procesem jest proces akwizycji, zajmuje się on zarządzaniem połączeniem z zestawem Neurosky MindWave Mobile (dalej w Pracy nazywany zestawem) oraz

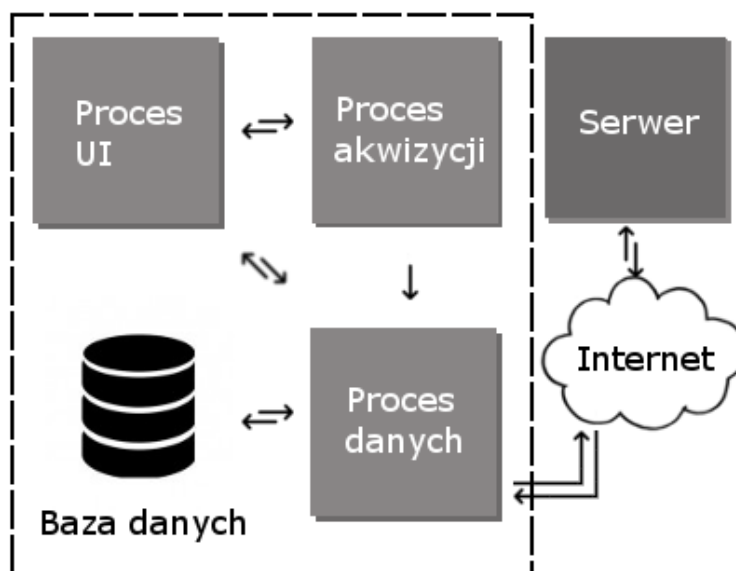
⁴ <http://archive.eclipse.org/eclipse/downloads/drops4/R-4.3.1-201309111000/>

⁵ <https://glassfish.java.net/download.html>

⁶ <http://developer.android.com/sdk/installing/studio.html>

⁷ Strona na której dostępne jest więcej informacji na temat pluginu i jego użycia
<https://github.com/jvoegele/gradle-android-plugin/wiki>

obsługą danych przychodzących z zestawu. Jest on uruchamiany za pośrednictwem obiektu klasy `EEGAcquisitionServiceConnectionConnector`.

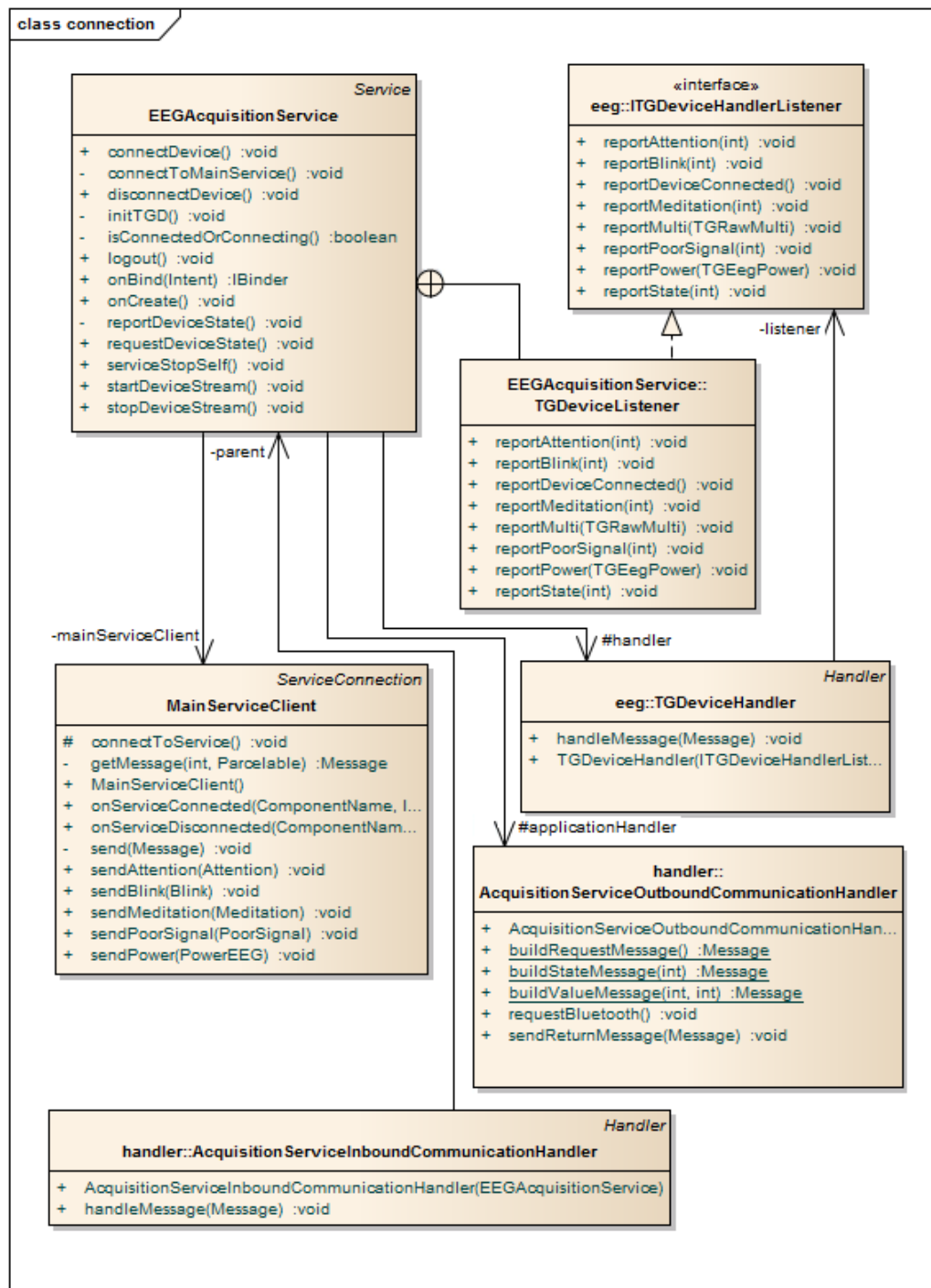


Rysunek 3-1 Podział aplikacji mobilnej na procesy

3.1.2 Struktura i działanie procesu akwizycji

Diagram klas przedstawiony na rysunku 3-2 pokazuje strukturę klas wykorzystywanych w procesie akwizycji. Główną klasą procesu jest `EEGAcquisitionService`. Odpowiedzialnością tej klasy jest koordynowanie komunikacji z procesem UI, procesem danych oraz zarządzanie połączeniem z urządzeniem. Pierwsza z odpowiedzialności jest realizowana przez parę klas `AcquisitionServiceInboundCommunicationHandler` oraz `AcquisitionServiceOutboundCommunicationHandler`, które zajmują się odpowiednio obsługą przyjmowanych komunikatów oraz tworzeniem i przesyłaniem wiadomości do procesu UI. Obiekty klasy odpowiadającej za przyjmowanie komunikatów dekodują wiadomości zgodnie z przyjętą reprezentacją (wykorzystano tutaj pola `arg1` oraz `arg2` obiektu klasy `Message`, bardziej szczegółowy opis sposobu komunikacji międzyprocesowej zawarto w rozdziale 3.1.4 Komunikacja międzyprocesowa) oraz wywołują odpowiednie metody klasy `EEGAcquisitionService`. Druga z wymienionych odpowiedzialności, czyli komunikacja (jednostronna) z procesem danych przypada na klasę `MainServiceClient`. Odpowiedzialnością tej klasy jest podpięcie (ang. *binding*) się do procesu danych, dokładniej do obiektu klasy `MainService`. Częścią zidentyfikowanych dla tej klasy odpowiedzialności jest budowanie i wysyłanie

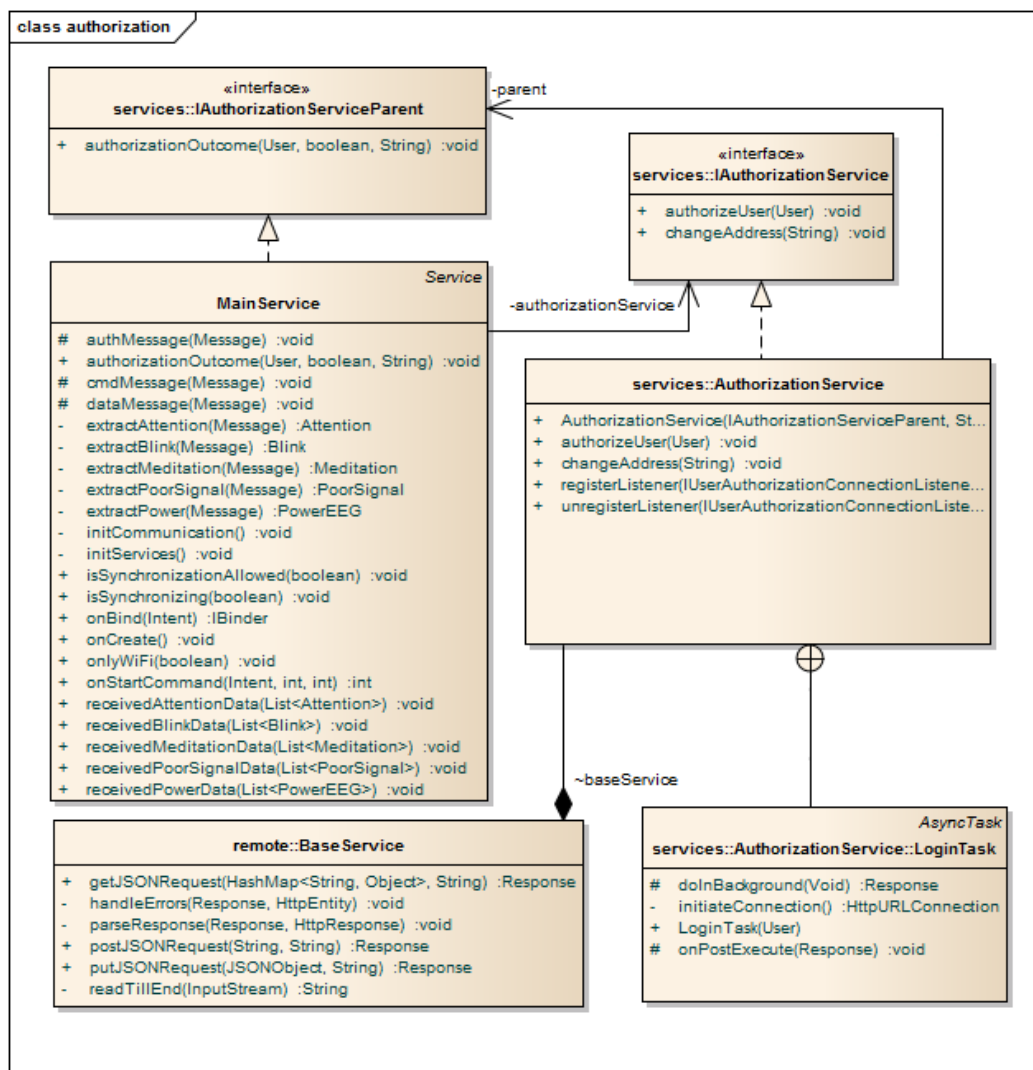
komunikatów zawierających aktualnie odbierane z zestawu dane. Ostatnie zadanie, które klasa ma wykonywać jest złożone z dwóch części – zarządzania połączeniem oraz odbierania danych emitowanych przez zestaw. To zadanie jest wypełniane przez klasę `TGDeviceHandler` poprzez interfejs `ITGDeviceHandlerListener` i klasę `TGDeviceListener` w połączeniu z klasą `TGDevice` dostarczaną razem z biblioteką obsługującą zestaw.



Rysunek 3-2 Struktura klas w procesie akwizycji danych

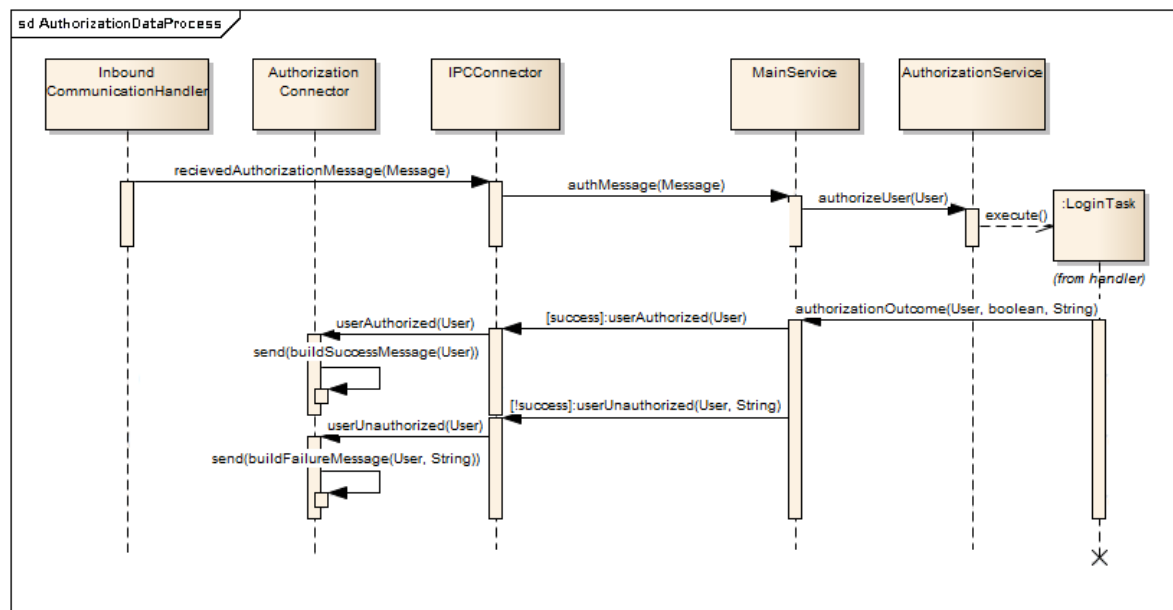
3.1.3 Struktura i działanie procesu danych

Struktura klas w procesie danych zostanie przedstawiona etapami, jako pierwsza omówiona zostanie część odpowiedzialna za autoryzację użytkownika. Rysunek 3-3 przedstawia strukturę klas biorących udział we wcześniej wymienionej czynności.



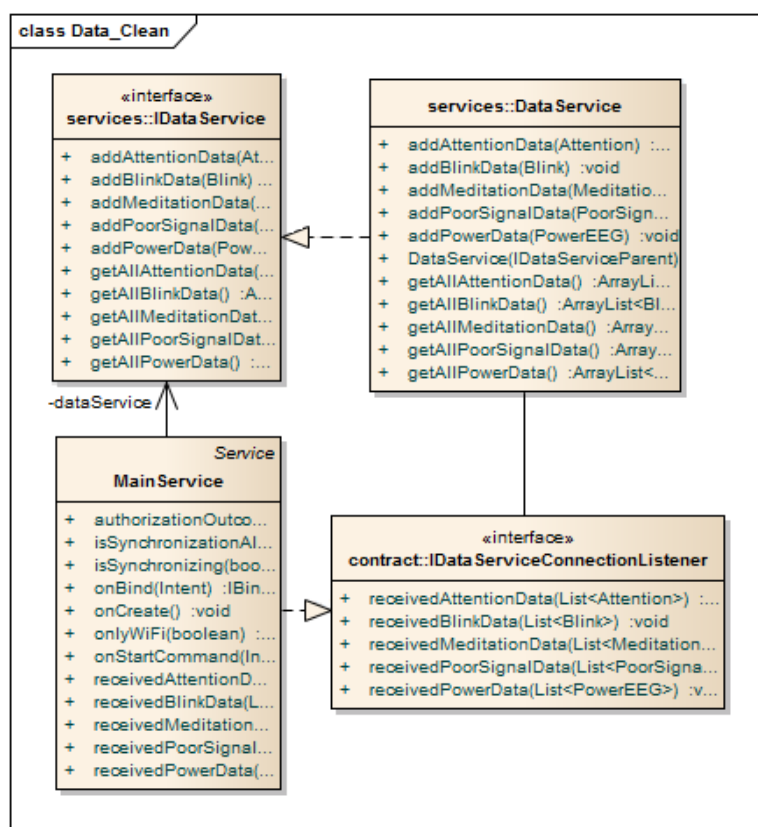
Rysunek 3-3 Struktura klas uczestniczących w autoryzacji użytkownika w procesie danych

Poniższy diagram sekwencji (rysunek 3-4) pokazuje, w jaki sposób zachodzi komunikacja między wymienionymi wyżej klasami oraz klasami pomocniczymi. Klucowym elementem wykonującym zapytania do serwera jest klasa `LoginTask`.



Rysunek 3-4 Diagram sekwencji pokazujący realizację zadania autoryzacji użytkownika wewnątrz procesu danych

Druga z omawianych części jest odpowiedzialna za obsługę danych. Musi ona umożliwić odczyt i zapis danych oraz odczyt danych od określonej daty. Ostatnia z wymienionych funkcjonalności jest konieczna, aby spełnić jedno z kluczowych założeń

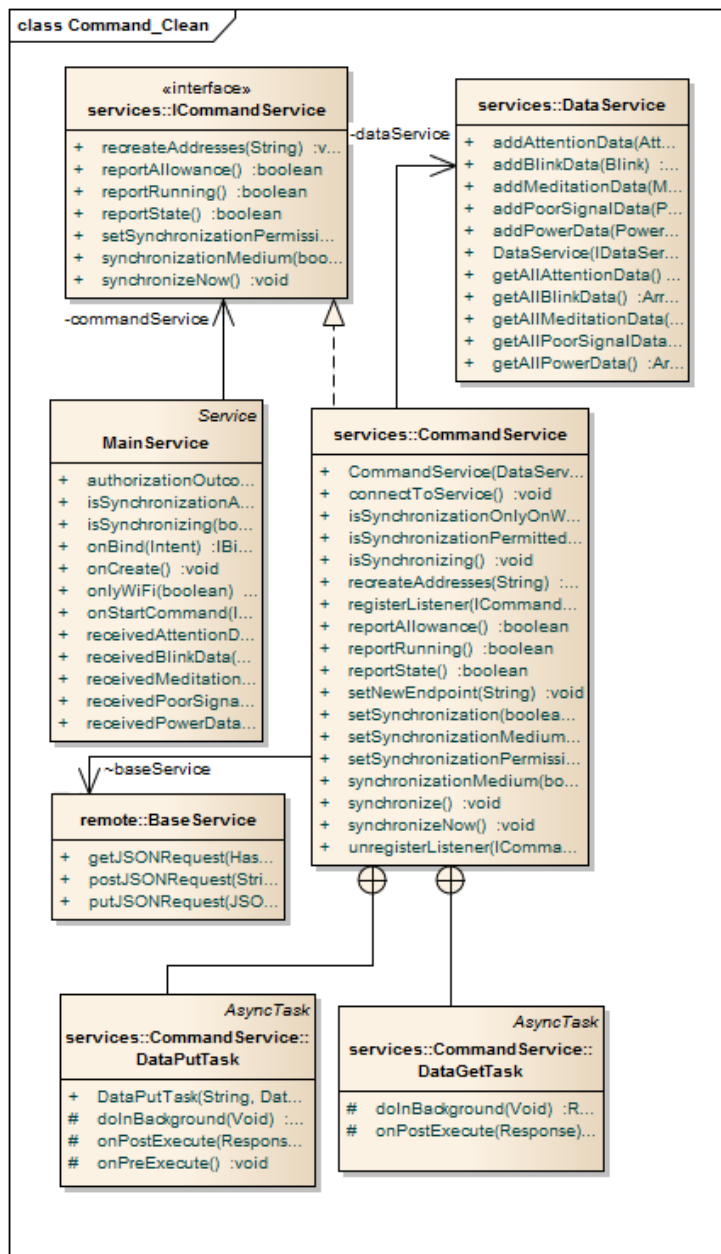


Rysunek 3-5 Struktura klas części procesu odpowiedzialnego za obsługę danych

funkcjonalnych aplikacji, wymaganie synchronizacji różnicowej. Może ona również posłużyć do wykonania usprawnień w zakresie interfejsu użytkownika, między innymi wyświetlania wykresów. Główne klasy biorące udział w tej czynności, które zostały napisane przez Autora, są wyszczególnione na rysunku 3-5.

Do odpowiedzialności ostatniej z części należy synchronizacja danych z serwerem oraz obsługa ograniczeń, które nałoży użytkownik. Zdecydowano, że ograniczenia, jakie użytkownik może nałożyć, są następujące:

- czy synchronizacja może nastąpić,
- czy synchronizacja może być wykonywana tylko gdy dostępne jest połączenie poprzez sieć bezprzewodową czy może zachodzić również przez sieć komórkową.



Rysunek 3-6 Klasy biorące udział w synchronizacji danych

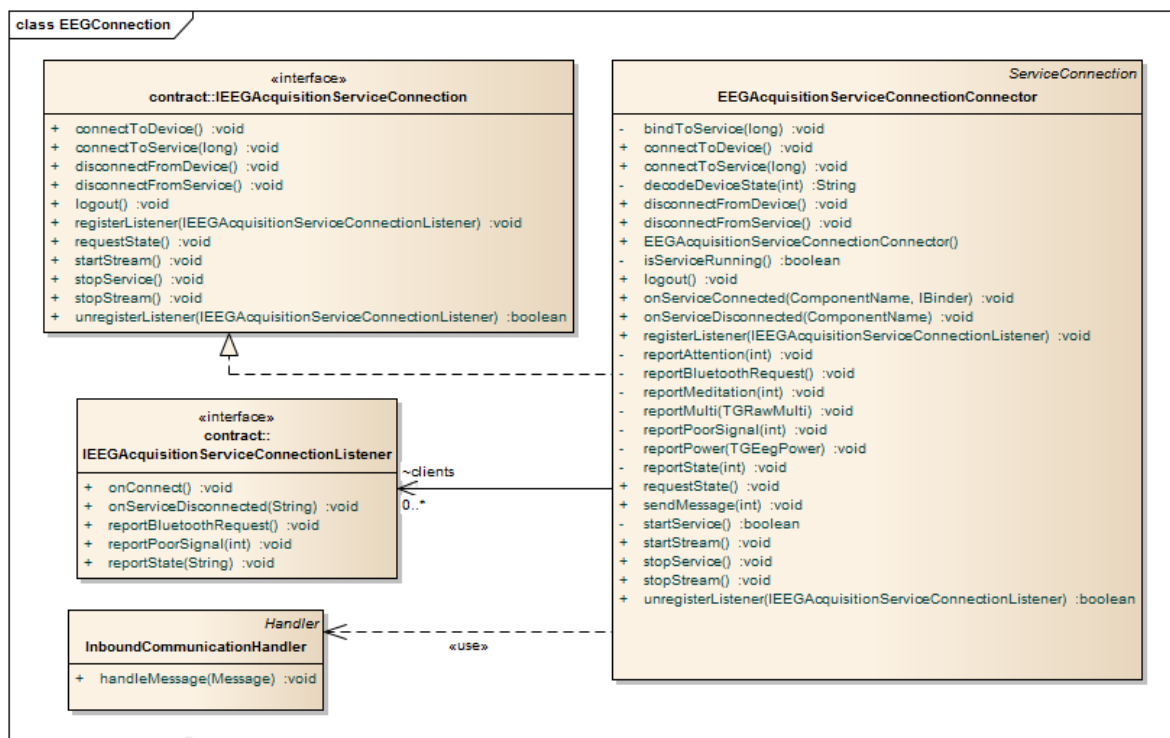
Na rysunku widocznym po lewej stronie (rysunek 3-6) ukazano główne klasy biorące udział w omawianym procesie.

Klasy `DataGetTask` oraz `DataPutTask` zajmują się komunikacją z serwerem. Pierwsza z nich jest odpowiedzialna za pobranie z serwera informacji o dacie ostatniej udanej synchronizacji dla danego użytkownika. Druga z wymienionych klas jest odpowiedzialna za wysyłanie danych do serwera. Rolę klasy `CommandService` sprowadzono w tym przypadku do zarządcy całego procesu wykorzystującego klasę `DataService` do odzyskania danych z lokalnej bazy danych oraz wcześniej omówionych klas do synchronizacji.

3.1.4 Komunikacja międzyprocesowa

Z powodu wybranej architektury aplikacji jej nieodzowną częścią jest komunikacja międzyprocesowa (z ang. *inter process communication (IPC)*). Procesy akwizycji oraz danych uruchamiane są w formie serwisów⁸ (ang. *service*). Przez wzgląd na dwa wspomniane powody zdecydowano się na implementację komunikacji z użyciem posłańca (z ang. *messenger*). W tym celu do każdego z procesów napisano osobne klasy realizujące komunikację.

Na poniższym rysunku przedstawiono klasy komunikujące się z procesem akwizycji.



Rysunek 3-7 Klasy wykonujące komunikację z procesem akwizycji

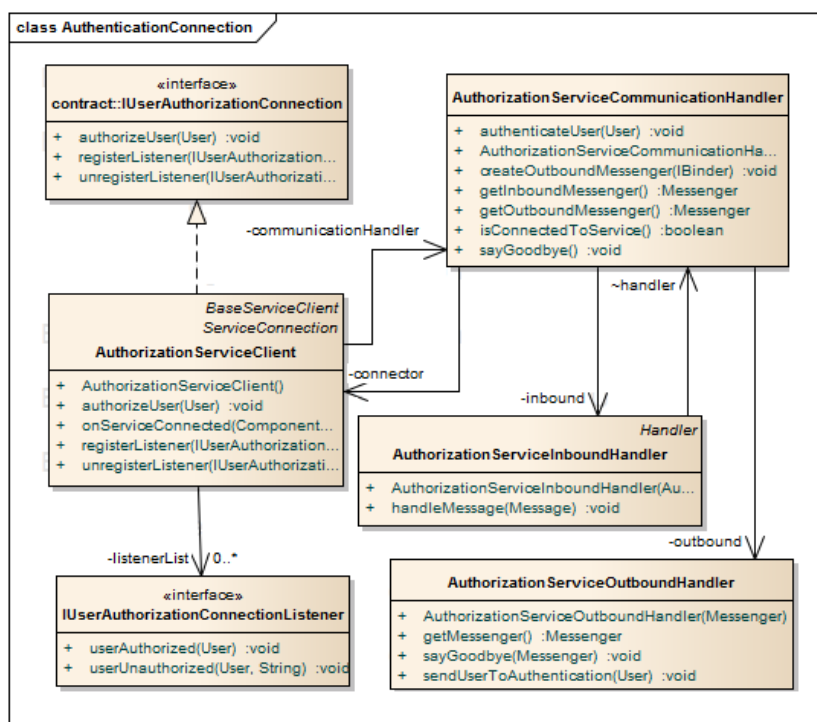
Strukturę aplikacji zorganizowano tak, aby komunikacja z danym procesem zachodziła przez łącznik – w tym opisywanym przypadku do serwisu akwizycji. Łącznik do poprawnego funkcjonowania wymaga, aby klasa z niego korzystająca (tylko gdy zachodzi potrzeba odbioru danych przez dany łącznik) implementowała interfejs pozwalający na komunikację zwrotną. Jednocześnie klasa łącznika (tutaj `EEGAcquisitionServiceConnectionConnector`) implementuje interfejs, poprzez który klienci łącznika mogą zwracać się do niego w celu wykonania określonego

⁸ Więcej na temat serwisów można przeczytać na stronie <http://developer.android.com/guide/components/services.html>

zadania. Wprowadzenie dodatkowego stopnia abstrakcji w postaci interfejsu pozwala na łatwą zmianę implementacji samego łącznika. Na przykład w momencie zmiany sposobu komunikacji na użycie interfejsów AIDL (Android Interface Definition Language⁹) jedyną zmianą, jaką trzeba będzie wprowadzić po stronie klienckiej, ograniczy się do dostarczenia nowej implementacji łącznika. Dodatkowym ułatwieniem jest założenie użycia w projekcie biblioteki do wstrzykiwania zależności, wtedy jedynym miejscem w którym należy zmienić przypisanie implementacji do interfejsu, jest moduł tej biblioteki. Tym samym zmiana tak ważnego fragmentu sprowadzona została do edycji dwóch miejsc.

Komunikacja z procesem danych została podzielona na trzy części, odpowiadające funkcjami logicznemu podziałowi zadań wprowadzonemu w rozdziale 3.1.3.

Jako pierwszy opisany został łącznik do celów autoryzacji. Poniżej widoczny jest diagram klas przedstawiający klasy wchodzące w skład łącznika oraz interfejsy wykorzystywane przy komunikacji z wcześniej wymienionym.

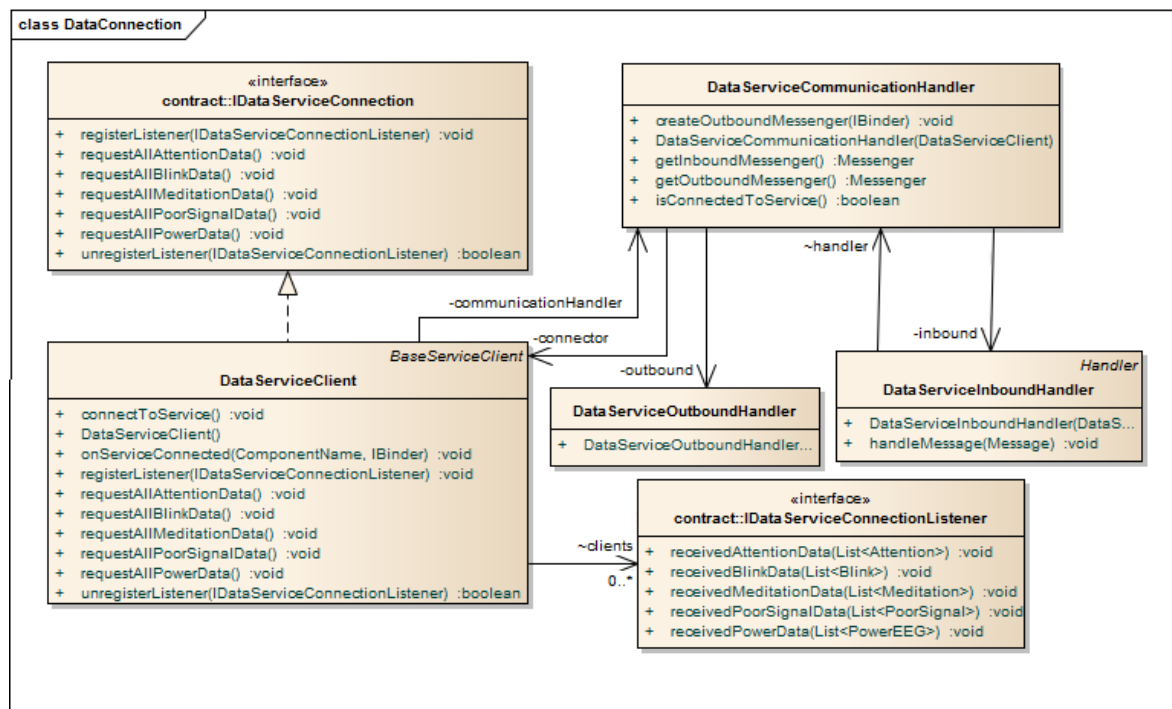


Rysunek 3-8 Łącznik udostępniający funkcję autoryzacji

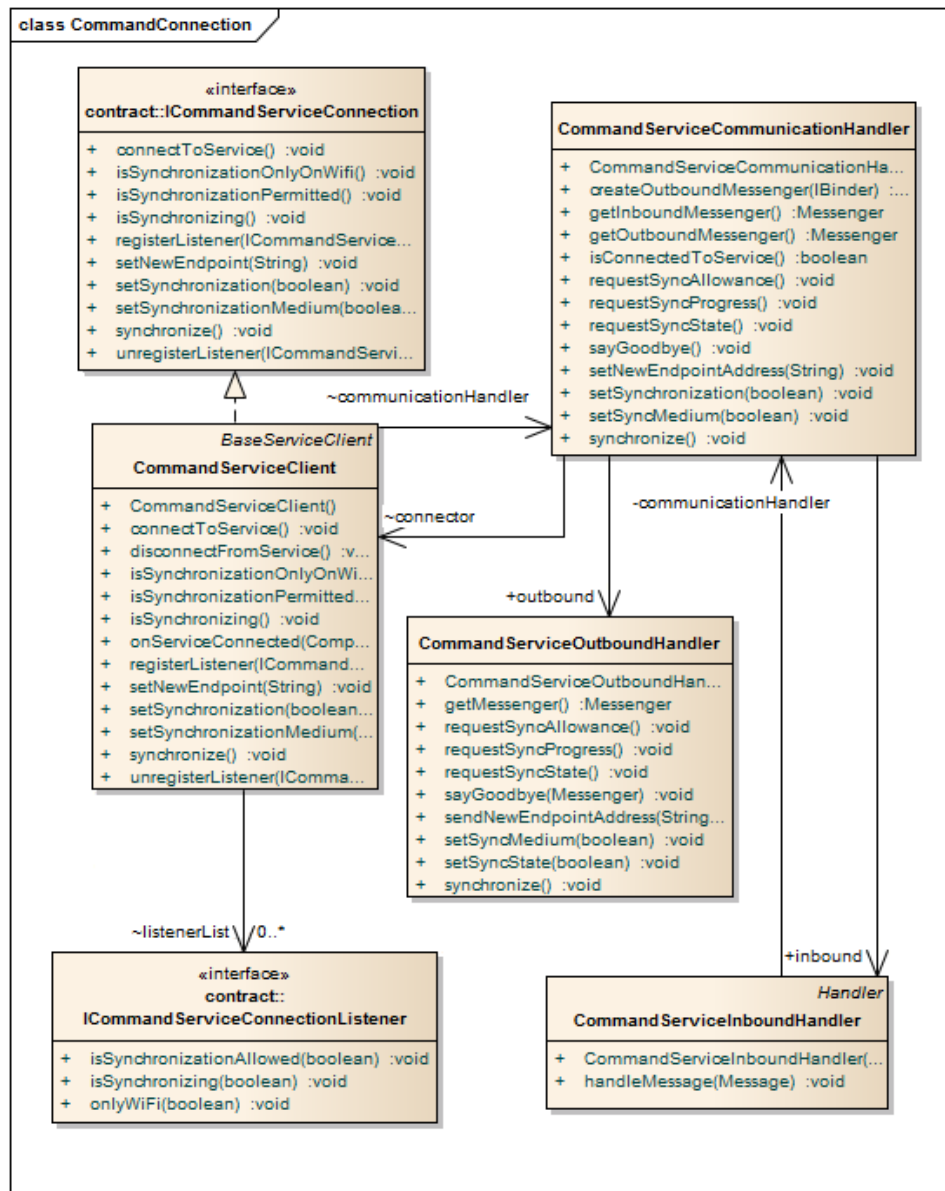
⁹ Na temat AIDL więcej informacji można zasięgnąć na stronie <http://developer.android.com/guide/components/aidl.html>

We wszystkich łącznikach komunikujących się z procesem danych wprowadzono dodatkowe klasy, `AuthorizationServiceCommunicationHandler`, `AuthorizationServiceInboundHandler` oraz `AuthorizationServiceOutboundHandler`. Dzięki tej modyfikacji udało się zredukować odpowiedzialności klasy `AuthorizationServiceClient` jedynie do zarządcy komunikacji, który określa co ma zostać wykonane, a samym wykonaniem zajmuje się klasa `AuthorizationServiceCommunicationHandler`, poprzez używane klasy `AuthorizationServiceOutboundHandler` lub `AuthorizationServiceInboundHandler`.

Analogicznie wyglądają łączniki danych oraz komend przedstawione odpowiednio na rysunku 3-9 oraz rysunku 3-10.



Rysunek 3-9 Klasy łącznika danych



Rysunek 3-10 Klasy łącznika komend

3.1.5 Organizacja bazy danych

Baza danych na potrzeby programu jest bardzo prosta, składa się z sześciu tabel, jednej przechowującej dane użytkowników oraz pięciu tabel przechowujących wartości wskaźników dostarczanych przez zestaw. Tabele z wartościami wskaźników utrzymują klucz obcy pozwalający powiązać daną wartość wskaźnika z konkretnym użytkownikiem, tym samym zapewniając wsparcie do korzystania z aplikacji przez wielu użytkowników.

3.2 Aplikacja serwerowa

3.2.1 Ogólny opis

Aplikacja serwerowa ma za zadanie udostępnić aplikacjom mobilnym możliwość zapisu danych do zewnętrznej bazy danych. Jednocześnie musi pozwalać na odczyt danych, które znajdują się już w bazie. Dodatkowo wymaga się, aby udostępniała możliwość weryfikacji danych użytkownika. Zaprojektowano ją tak, aby była zgodna z opisanym w rozdziale 2.1 wzorcem REST.

Ogólna struktura tej aplikacji przedstawiona jest na Rysunek 2-4.

3.2.2 Udostępniane punkty końcowe

Aplikacja udostępnia ogółem pięć funkcjonalnych punktów końcowych (ang. *endpoint*).

3.2.2.1 Logowanie

Najpierw opisany zostanie punkt udostępniający funkcjonalność logowania. Jest on dostępny pod adresem relatywnym do adresu web serwisu `/login`. Wspiera on tylko zapytania wysyłane metodą POST. Serwlet obsługujący działanie tego punktu końcowego oczekuje, że w ciele zapytania (ang. *request body*) znajdzie się obiekt JSON. Struktura tego obiektu przedstawiona jest na listingu poniżej:

```
{
    id: ID_użytkownika ,
    password: hasło_użytkownika
}
```

Listing 3-1 Format zapytania weryfikującego tożsamość użytkownika

W celu udzielenia odpowiedzi używane jest pole statusu nagłówka http odpowiedzi, którego wartość ustawiana jest na 200 (ok), gdy autoryzacja przebiegła pomyślnie oraz 401 (nieautoryzowany z ang. *unauthorized*), gdy nie powiodła się (dodatkowo przekazywany jest powód odrzucenia).

3.2.2.2 Synchronizacja różnicowa

Funkcjonalność zapisywania danych z użyciem metody różnicowej wymaga udostępnienia ostatniej daty akwizycji przetrzymywanej w bazie dla każdego z parametrów. Zadanie to realizowane jest przez serwlet `DataLastUpdateServlet` odpowiadający za punkt końcowy `/data/last`.

Serwlet ten oczekuje, że zapytanie skierowane do niego realizowane będzie za pomocą metody GET, oraz obecny będzie parametr `userID` odpowiadający identyfikatorowi użytkownika. W odpowiedzi zwrócony zostanie obiekt JSON o następującym formacie:

```
{
    "attention" : czas w milisekundach,
    "meditation" : czas w milisekundach,
    "blink" : czas w milisekundach,
    "power" : czas w milisekundach,
    "poorSignal" : czas w milisekundach
}
```

Listing 3-2 Format odpowiedzi zawierającej ostatnie czasy aktualizacji

3.2.2.3 Zapisywanie danych

Punkt końcowy udostępniający możliwość zapisywania danych do głównej bazy danych dostępny jest pod adresem `/data`. Obsługą tego punktu zajmuje się klasa `DataServlet`, która przy użyciu metody PUT oczekuje na przekazanie przez ciało zapytania obiektu JSON, który będzie zgodny z następującym formatem:

```
{
    "attentions" : [obiekt wskaźnika],
    "meditations" : [obiekt wskaźnika],
    "blinks" : [obiekt wskaźnika],
    "powers" : [obiekt złożonego wskaźnika],
    "signals" : [obiekt wskaźnika]
}
```

Listing 3-3 Format danych przesyłanych do serwera w celu utrwalenia

Każde z pól w powyższym obiekcie jest tablicą określonych obiektów, które opisane zostały poniżej.

```
{
    "user" : identyfikator użytkownika,
    "value" : wartość wskaźnika,
    "date" : czas w milisekundach
}
```

Listing 3-4 Obiekt JSON reprezentujący wartości prostego wskaźnika

Obiekt z Listing 3-4 przechowuje wartości wskaźników prostych (posiadających tylko jedną wartość). Odpowiada przekazywaniu obiektów wszystkich wskaźników poza `PowerEEG`, który wymaga innej reprezentacji (ponieważ zawiera w sobie wartości każdej z ośmiu opisanych w rozdziale 2.2 fal jednocześnie).

Struktura obiektu JSON używanego do przekazywania obiektów klasy `PowerEEG` jest widoczna na Listing 3-5, widocznym poniżej.

```
{
    "user" : identyfikator użytkownika,
    "lowAlpha" : wartość wskaźnika lowAlpha,
    "highAlpha" : wartość wskaźnika highAlpha,
    "lowBeta" : wartość wskaźnika lowBeta,
    "highBeta" : wartość wskaźnika highBeta,
    "lowGamma" : wartość wskaźnika lowGamma,
    "midGamma" : wartość wskaźnika midGamma,
    "theta" : wartość wskaźnika theta,
    "delta" : wartość wskaźnika delta,
    "date" : data w milisekundach
}
```

Listing 3-5 Obiekt JSON reprezentujący wartości złożonego wskaźnika

Serwlet w odpowiedzi na żądanie zapisu zwraca nagłówek `http` z odpowiednim kodem statusu (ang. *status code*) oraz komunikatem przekazany jako ciało odpowiedzi (ang. *response body*). Wartość pola statusu ustawiana jest na 200, gdy wszystkie operacje wykonane zostały poprawnie, 400 w przypadku złego formatu danych lub 500, gdy wystąpi błąd zapisu do bazy danych.

3.2.2.4 Weryfikacja serwisu

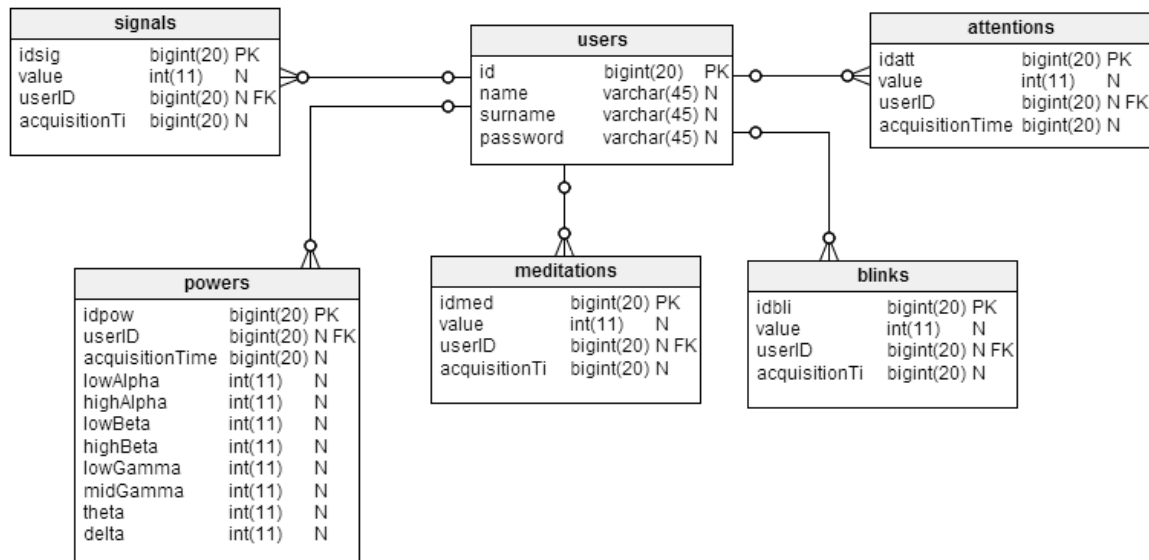
W celu ułatwienia weryfikacji adresu podanego przez użytkownika (wprowadzanego ręcznie w aplikacji mobilnej) przygotowano punkt końcowy, który na każde zapytanie GET lub POST odsyła odpowiedź z nagłówkiem `http`, którego pole statusu jest ustawiane na wartość 200. Ten punkt końcowy jest obsługiwany przez klasę `CheckServlet` i dostępny pod adresem `/check`.

3.2.2.5 Podgląd danych

Jednym z wymagań stawianych aplikacji serwerowej jest możliwość przeglądania danych. Udostępniono ją za pośrednictwem punktu końcowego znajdującego się pod adresem `/data/view` i obsługiwanego poprzez klasę `DataViewServlet` reagującą na zapytania wysyłane metodą POST. Klasa obsługuje trzy akcje: przeprowadza proces logowania osoby uprzywilejowanej do przeglądania bazy danych, obsługuje zlecenia dodania osoby do bazy oraz wyświetla dane dotyczące danego użytkownika. Decyzja o wykonaniu jednej z wymienionych funkcji podejmowana jest na podstawie wartości parametru `action` przekazywanego w nagłówku.

3.2.3 Organizacja bazy danych

Baza danych odpowiadająca potrzebom aplikacji jest bardzo prosta, składa się z sześciu tabel. Główną tabelą jest tabela `users`, przechowująca podstawowe informacje dotyczące użytkowników. Pozostałe pięć tabel posiada powiązania z tabelą główną (w celu ustalenia do którego użytkownika należą dane) i przechowuje wartości odpowiednich wskaźników oraz czasy ich akwizycji.



Rysunek 3-11 Schemat bazy danych

4 Implementacja

4.1 Aplikacja mobilna

4.1.1 Użyte biblioteki

Obecnie budowanie aplikacji bez użycia zewnętrznych bibliotek jest nie tyle co niemożliwe a niepraktyczne i zbędne. Biblioteki dostarczają gotowych rozwiązań najczęściej występujących, powtarzających się problemów, pomagają w rozwiązywaniu już istniejących lub zwyczajnie ułatwiają pracę programiście.

W projekcie zastosowano kilka bibliotek, pierwszą z opisanych będzie stworzony przez Jake’a Whartona Butterknife¹⁰. Biblioteka ta służy zwiększeniu czytelności (głównie) klas aktywności (ang. *activity*) przez zastąpienie kodu, który służy przygotowaniu obiektów obecnych w widoku do użycia. Wykonywane jest to za pomocą adnotacji, które przed kompilacją zamieniane są na normalny kod, w związku z tym nie wpływa to negatywnie na działanie programu.

Drugą z bibliotek, które zostały użyte, jest Dagger¹¹ stworzony przez firmę Square. Biblioteka ta jest lekkim (ang. *lightweight*) i szybkim frameworkiem do wstrzykiwania zależności bazującym na adnotacjach `javax.inject`¹²(JSR-330). W prostych przypadkach Dagger bazuje swoje działanie na użyciu domyślnych, bezparametrowych konstruktorów. W bardziej skomplikowanych przypadkach (gdzie występują parametry lub modyfikacja kodu jest zwyczajnie niemożliwa) używa się tak zwanych modułów, w których definiuje się metody dekorowane adnotacją `@Provides` konfiguruje instancję klasy. Dodatkowo nie wymusza dziedziczenia po jakiegokolwiek klasie, jak ma to miejsce w innych frameworkach (RoboGuice¹³). Jego możliwości są o wiele większe aniżeli opisano w tym tekście, aby dowiedzieć się więcej, warto zapoznać się z nimi na stronie projektu¹³.

¹⁰ Strona projektu znajduje się pod adresem <http://jakewharton.github.io/butterknife/>, zawiera opis i kompletne wprowadzenie do użycia.

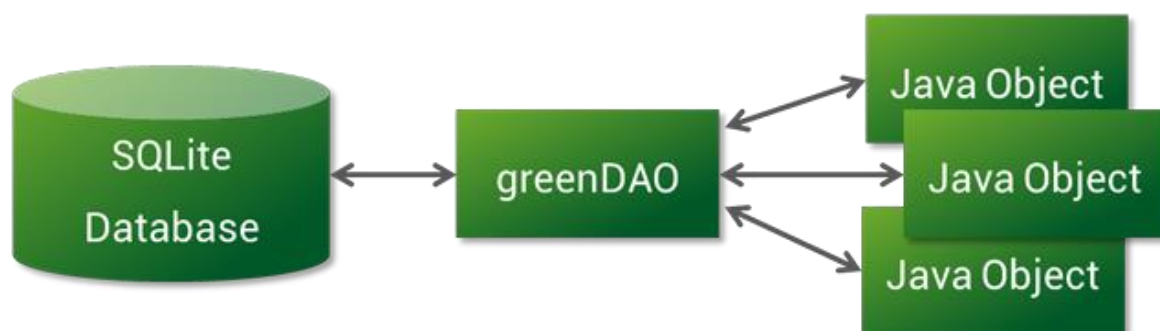
¹¹ Strona domowa projektu może zostać znaleziona pod adresem <http://square.github.io/dagger/> i również zawiera wprowadzenie do użycia biblioteki.

¹² Specyfikacja dostępna pod adresem <https://jcp.org/aboutJava/communityprocess/final/jsr330/>

¹³ Strona projektu <https://code.google.com/p/roboguice/>

SlidingMenu¹⁴ autorstwa Jeremiego Feinsteina jest ostatnią z bibliotek wspomagających tworzenie interfejsu użytkownika użytych w projekcie. Pozwala ona, na łatwe zastosowanie tak zwanego przesuwanego menu (z ang. *sliding menu*). Menu tworzone z użyciem biblioteki może być dostosowywane do potrzeb poprzez liczne parametry (jak długość otwierającego menu ruchu palcem, stronę z której jest pokazywane etc.).

W celu ułatwienia manipulacji danymi w wewnętrznej bazie SQLite użyto biblioteki mapującej encje z bazy danych na obiekty Javy (mapowanie obiektowo-relacyjne, ORM), greenDAO¹⁵. Dzięki niej z programisty zostaje zdjęty obowiązek utrzymywania wszystkich zapytań i konwersji obiektów na encje oraz encji na obiekty.



Rysunek 4-1 Zasada działania biblioteki greenDAO, źródło strona projektu¹⁵

Ostatnią biblioteką użytą w projekcie jest ThinkGear¹⁶, jest to biblioteka będąca częścią narzędzi deweloperskich w wersji 3 dla platformy Android. Pozwala ona na komunikację z urządzeniem Neurosky MindWave Mobile używanym w projekcie do akwizycji sygnałów EEG.

4.1.2 Szczegóły implementacji

Na następnej stronie widoczny jest nagłówek klasy `MainService` zajmującej się obsługą danych, dokładniej buforowaniem ich w lokalnej bazie i przesyłaniem do serwera oraz autoryzacją użytkowników.

Klasa ta będąc główną klasą procesu rozszerza dostarczaną z systemem Android klasę `Service` w celu uzyskania dostępu do metod cyklu życia serwisu (ang. *service*). Zaimplementowano w niej również dwa interfejsy służące obsłudze komunikatów

¹⁴ Strona domowa projektu <https://github.com/jfeinstein10/SlidingMenu>

¹⁵ Strona domowa projektu zawierająca liczne wprowadzenia i dokumentację znajduje się pod adresem <http://greendao-orm.com/documentation/introduction/>

¹⁶ Do pobrania za darmo ze strony <http://store.neurosky.com/products/developer-tools-3-android>

zwrotnych z agregowanych obiektów klas pełniących specjalizowane role. Jako że w ramach całej aplikacji istnieje tylko jeden obiekt klasy, logicznym było zaimplementowanie jej jako wzorzec Singleton z użyciem adnotacji dostępnej w specyfikacji JSR-330.

```

1: @Singleton
2: public class MainService extends Service
3: implements IAuthorizationServiceParent,
   IDataServiceParent {
4:
5:     private static final String TAG =
        MainService.class.getSimpleName();
6:     private Messenger messenger;
7:     private IPCCConnector mainConnector;
8:     private ICommandService commandService;
9:     private IAuthorizationService authorizationService;
10:    private IDataService dataService;
11:    private String endpointAddress;
12:
13:    private void initServices() {
14:        dataService = new DataService(this);
15:        authorizationService =
            new AuthorizationService(this, endpointAddress);
16:        commandService =
            new CommandService((DataService) dataService);
17:    }
18:
19:    private Attention extractAttention(Message msg) {
20:        Bundle bundle = msg.getData();
21:        if(bundle != null){
22:            bundle.setClassLoader(
                Attention.class.getClassLoader());
23:            Attention data = bundle.
                getParcelable(IPCCConnector.DATA_DATA);
24:            return data;
25:        } else {
26:            Log.i(TAG,
                "Attention has not been passed with bundle! ");
27:            return null;
28:        }
29:    }
...
237: }

```

Listing 4-1 Nagłówek i pola klasy MainService

Klasa jest logicznie podzielona na cztery części, dokładnie przedstawione w 3.1.3. Zaczynająca się w linii 13 powyższego listingu metoda `initServices()` umożliwia prawidłowe działanie całej aplikacji poprzez inicjalizację głównych komponentów obiektu.

Wartym wyszczególnienia jest rozwiązanie problemu przekazywania niestandardowych obiektów poprzez granicę międzyprocesową. Niemożliwym jest przekazanie takiej klasy bez specjalnych operacji. Obiekty w celu przeniknięcia wspomnianej bariery muszą zostać poddane serializacji (ang. *serialize*), a ładowacz klas (ang. *class loader*) dokonujący deserializacji (ang. *deserialize*) z racji znajdowania się w innym procesie, nie wie o istnieniu wcześniej wspomnianej klasy. Z tego powodu należy do obiektu `Bundle` przypisać odpowiedni ładowacz klas przed deserializacją żadanego obiektu. Operacja ta widoczna jest w linii 22 Listing 4-1.

Przedstawione na Listing 4-2 wybrane fragmenty kodu należą do klasy `EEGAcquisitionService`, będącej główną klasą procesu akwizycji danych. Z tego powodu klasa ta, również rozszerza systemową klasę `Service`.

Metoda wywoływana jako pierwsza z cyklu życia obiektu rozszerzającego klasę `Service` (pomijając konstruktor) jest `onCreate()`, widoczna w linii 19. Zadaniem tej metody jest skonfigurowanie instancji klasy tak, aby mogła działać poprawnie. W tym przypadku metoda tworzy obiekt klasy `Messenger` inicjując go również nowotworzonym obiektem klasy rozszerzającej systemowy `Handler`. Wywołuje również metodę inicjującą obiekty wymagane do połączenia z zestawem Neurosky oraz metodę ustanawiającą połączenie z procesem danych. Ważnym jest tutaj, że `onCreate()`, podobnie jak konstruktor, wywoływana jest tylko raz dla każdego obiektu.

Konieczną do opisanego z punktu widzenia komunikacji międzyprocesowej jest metoda `onBind()`, która wywoływana jest każdorazowo przy podłączeniu do procesu za pomocą metody `bindService()` klasy `Context`. Do metody przekazywany jest obiekt klasy `Intent` zawierający w sobie obiekt klasy `Bundle`, który z kolei zawiera obiekt klasy `Messenger` potrzebny do realizacji dwukierunkowej komunikacji. Zwraca natomiast obiekt implementujący interfejs `IBinder` wykorzystywany do stworzenia obiektu `Messenger` po drugiej stronie połączenia.

```

1:  public class EEGAcquisitionService extends Service {
2:  //dla zwiększenia czytelności pominięto definicje stałych
3:      private long userId;
4:      protected Messenger messenger;
5:      protected BluetoothAdapter btAdapter;
6:      protected TGDevice tgDevice;
7:      protected TGDeviceHandler handler;
8:      protected AcquisitionServiceOutboundCommunicationHandler
           applicationHandler;
9:      private MainServiceClient mainServiceClient;
10:
11:      @Override
12:      public IBinder onBind(Intent intent) {
13:          Messenger returnMessenger = (Messenger) intent
               .getParcelableExtra("Messenger");
14:          userId = intent
               .getLongExtra("UserID", Long.MIN_VALUE);
15:          applicationHandler = new
AcquisitionServiceOutboundCommunicationHandler(returnMessenger);
16:          return messenger.getBinder();
17:      }
18:
19:      @Override
20:      public void onCreate() {
21:          super.onCreate();
22:          messenger = new Messenger(new
AcquisitionServiceInboundCommunicationHandler(this));
23:          initTGD();
24:          connectToMainService();
25:      }
26:
27:      private void connectToMainService() {
28:          mainServiceClient = new MainServiceClient();
29:          mainServiceClient.connectToService();
30:      }
31:
32:      private void initTGD() {
33:          btAdapter = BluetoothAdapter.getDefaultAdapter();
34:          handler =
               new TGDeviceHandler(new TGDeviceListener());
35:          tgDevice = new TGDevice(btAdapter, handler);
36:      }
...
199: }

```

Listing 4-2 Wybrane fragmenty klasy EEGAcquisitionService

4.2 Aplikacja serwerowa

Do tworzenia aplikacji wybrano środowisko programistyczne Eclipse w wersji Kepler oraz serwer Glassfish w wersji 4.0.

4.2.1 Użyte biblioteki

Jednym z głównych zadań aplikacji serwerowej jest komunikacja z bazą danych. Aby umożliwić realizację tego zadania należało skorzystać z biblioteki MySQL Connector¹⁷ dla języka Java.

Użyto również dwóch bibliotek z projektu Apache Commons¹⁸, dokładnie Commons IO oraz Commons Lang. Pierwsza z bibliotek służy do usprawnienia obsługi operacji wejścia/wyjścia. Druga służy do usprawnienia posługiwania się ciągami znaków, oferując wiele użytecznych metod niedostępnych w podstawowej implementacji API Javy. Ponieważ komunikacja pomiędzy klientami a serwerem zachodzi z użyciem formatu JSON, wymusiło to użycie biblioteki JSON-java¹⁹ do obsługi przesyłanych wiadomości.

4.2.2 Szczegóły implementacji

Komunikacją z bazą danych zajmują się trzy klasy `DBAccessor`, `DBWriter` oraz `DBReader` z pakietu `pl.mbos.bachelor_thesis.db`. Pierwsza z wymienionych klas odpowiada za uzyskanie dostępu do bazy danych, pozostałe dwie odpowiednio za obsługę zapisu oraz odczytu danych z bazy danych. Wprowadzenie dodatkowej warstwy abstrakcji pozwala na utrzymanie całej logiki dostępu w jednym pakiecie.

Na Listing 4-3 widoczna jest metoda klasy `DBAccessor` udostępniająca połączenie do bazy danych. Parametry używane do zestawienia połączenia zdefiniowane są jako prywatne właściwości klasy.

Fragment klasy `DBReader` z przykładową metodą odczytującą dane dotyczące wskaźnika skupienia z bazy danych. Metody odczytujące wartości pozostałych wskaźników różnią się jedynie zwracanym typem oraz zapytaniem widocznym w 8 linii na Listing 4-4.

¹⁷ Dostępny do pobrania ze strony projektu: <http://dev.mysql.com/downloads/connector/j/>

¹⁸ Strona projektu dostępna pod adresem <http://commons.apache.org/>

¹⁹ Źródła biblioteki znajdują się na stronie projektu <http://www.json.org/java/>, biblioteka dostępna jest z repozytorium maven central pod adresem <http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22org.json%22>

```

1: public static Connection getConnection()
2: throws SQLException {
3:     Connection conn = null;
4:     Properties connectionProps = new Properties();
5:     connectionProps.put("user", userName);
6:     connectionProps.put("password", password);
7:     conn = DriverManager.getConnection("jdbc:mysql://" +
        serverAddress + ":" + portNumber + "/", connectionProps);
8:     System.out.println("Connected to database");
9:     return conn;
10: }

```

Listing 4-3 Metoda udostępniająca połączenie z bazą danych

```

1: public class DBReader {
2:     private String dbSchema = "application.";
3:     public List<Attention> getAttentionForUser(
        Long userID) {
4:         List<Attention> attentions = null;
5:         try {
6:             Connection connection = DBAccessor.
                getConnection();
7:             Statement stmt = connection.createStatement();
8:             ResultSet result = stmt.
                executeQuery("SELECT * FROM " + dbSchema + "attentions
                    WHERE userID=" + userID.toString());
9:             if (result.isBeforeFirst()) {
10:                 attentions =
                    new ArrayList<Attention>();
11:                 while (result.next()) {
12:                     Attention na =
                        new Attention(result.getLong("userID"),
                            result.getInt("value"),
                            new Date(result.getLong("acquisitionTime")));
13:                     attentions.add(na);
14:                 }
15:             }
16:             connection.close();
17:         } catch (SQLException e) {
18:             e.printStackTrace();
19:         }
20:         return attentions;
21:     }
...
208: }

```

Listing 4-4 Fragment klasy dokonującej odczytu z bazy danych

```
1: public class DBWriter {
2:     private static final String SCHEMA = "application";
3:
4:     public void persistData(Container dataContainer,
5:                             OperationResult result) {
6:         try {
7:             persistAttention(dataContainer.getAttentions());
8:             persistMeditation(dataContainer.getMeditations());
9:             persistBlink(dataContainer.getBlinks());
10:            persistPowers(dataContainer.getPowers());
11:            persistSignals(dataContainer.getSignals());
12:        } catch (SQLException e) {
13:            result.outcome = e.getMessage();
14:            result.code =
15:                HttpServletResponse.SC_INTERNAL_SERVER_ERROR;
16:        }
17:    }
18:
19:     private void persistAttention(List<Attention> signals)
20:         throws SQLException {
21:         Connection connection =
22:             DBAccessor.getConnection();
23:         Statement createStatement =
24:             connection.createStatement();
25:         for (Attention att : signals) {
26:             String sql = "INSERT INTO `"
27: + SCHEMA + "`.`attentions` (value,userID,acquisitionTime)"
28: + "VALUES(" + att.getValue() + "," + att.getUserID() + ","
29: + att.getCollectionDate().getTime() + ")" ";
30:             createStatement.addBatch(sql);
31:         }
32:         createStatement.executeBatch();
33:         connection.close();
34:     }
35: }
```

Listing 4-5 Fragment klasy zapisującej dane do bazy danych

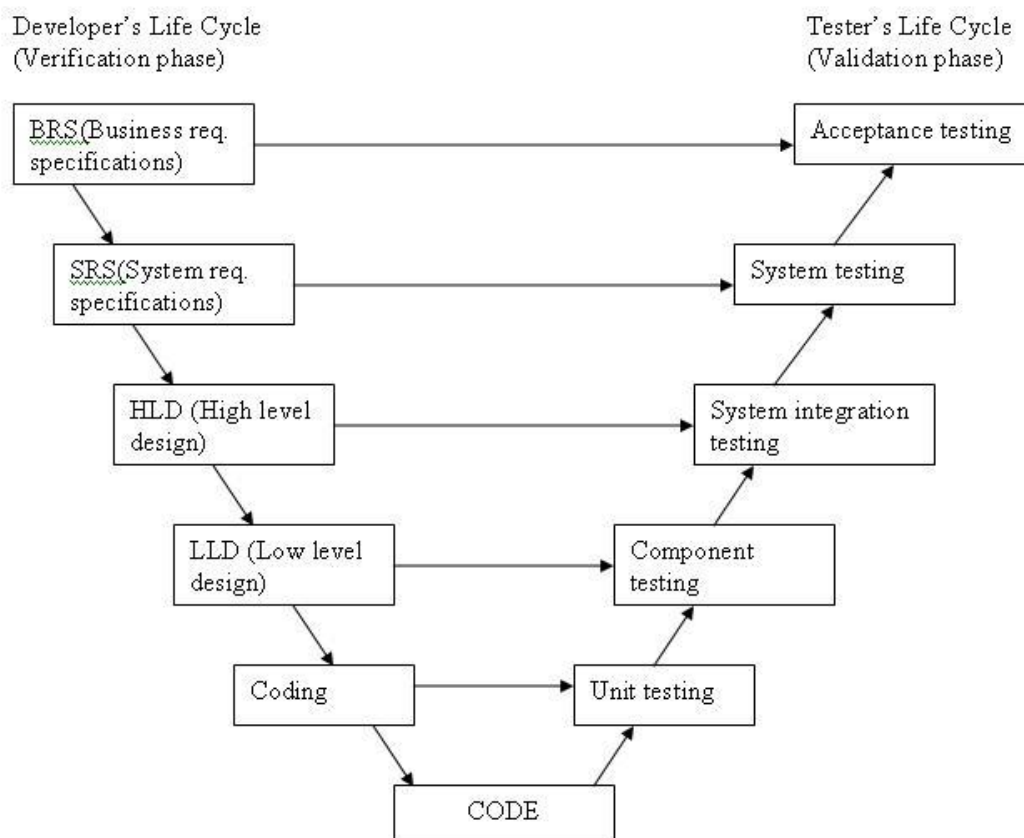
Fragment klasy DBWriter przedstawiony na Listing 4-5 zawiera główną metodę używaną do zapisu paczki danych otrzymanych od klienta (`persistData()`). Jej działanie polega na wywołaniu prywatnych metod klasy w celu zapisu danych dotyczących poszczególnych wskaźników. Jako przykład metody zapisującej podano metodę `persistAttentions()`, która zapisuje dane o wskaźniku skupienia.

Zapis każdego zestawu danych (wartości jednego ze wskaźników) wykonywany jest jako jedna operacja wsadowa, pozwala to przyspieszyć operacje na bazie danych.

5 Testowanie i uruchamianie

5.1 Model wzorcowy

Mając na uwadze to, że wymagania stawiane aplikacji są zrozumiałe i nie powinny zmieniać się przez czas życia projektu oraz to, że sam projekt nie jest duży zdecydowano się na prowadzenie go kierując się modelem V²⁰.



Rysunek 5-1 Wizualizacja modelu V, źródło²¹

W małych projektach, podobnych do wykonywanego przez Autora, model ten powinien sprawdzić się bardzo dobrze.

²⁰ Artykuł przybliżający specyfikę modelu V dostępny jest pod adresem <http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/> [dostęp 13 stycznia 2014]

²¹ http://www.the-software-experts.de/e_dta-sw-process.htm

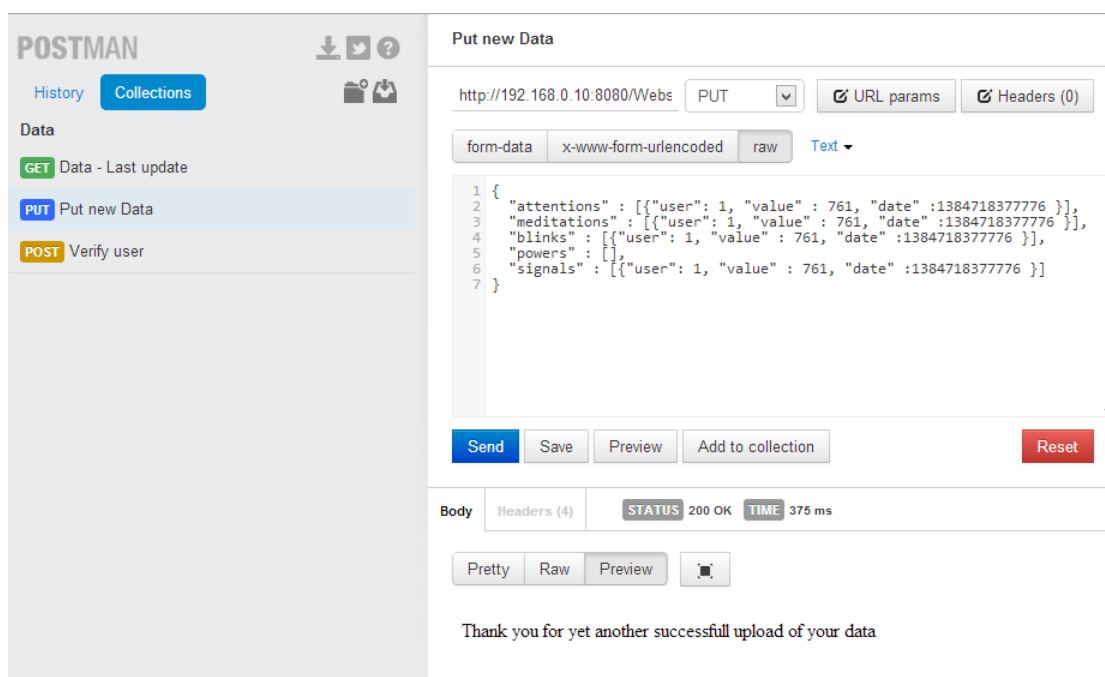
5.2 Testowanie aplikacji mobilnej

Projekt zdecydowano rozpocząć od implementacji części mobilnej. Podczas każdego z etapów powstawania tej części pracy na bieżąco wykonywano testy. Rozpoczynając od procesu analizy i projektowania ogólnej struktury aplikacji, gdzie testami były ręczne przejścia przez szkicowane na papierze encje reprezentujące poszczególne części systemu a kończąc na fazie implementacji interfejsu użytkownika, gdzie testowanie i uruchamianie odbywało się przy podejściu czarnej skrzynki (ang. *black box testing*).

5.3 Testowanie aplikacji serwerowej

Aplikacja serwerowa przez wzgląd na prostotę została wykonana z wykorzystaniem jedynie testowania akceptacyjnego. W celu przesyłania spreparowanych zapytań mających aktywować odpowiednie funkcje serwisu użyto programu Postman²².

Na kolejnych stronach przedstawiono kilka zapytań z serii testów akceptacyjnych interfejsów udostępnianych aplikacji mobilnej, generujących tak pozytywne, jak i negatywne przypadki.



Rysunek 5-2 Zrzut ekranu pokazujący program użyty do testowania części serwerowej

²² Strona projektu dostępna w sieci pod adresem <http://www.getpostman.com/>

Verify user

http://192.168.0.10:8080/Webservice/login POST [URL params](#) [Headers \(0\)](#)

[form-data](#) [x-www-form-urlencoded](#) [raw](#) [Text](#)

```
1 {  
2   "id": "1",  
3   "password": "4"  
4 }
```

[Send](#) [Save](#) [Preview](#) [Add to collection](#) [Reset](#)

Body [Headers \(4\)](#) **STATUS** 401 Unauthorized **TIME** 78 ms

[Pretty](#) [Raw](#) [Preview](#) [JSON](#) [XML](#)

```
1 Incorrect password, try again  
2
```

Rysunek 5-3 Odpowiedź serwera w przypadku braku autoryzacji (podane hasło jest niepoprawne)

Verify user

http://192.168.0.10:8080/Webservice/login POST [URL params](#) [Headers \(0\)](#)

[form-data](#) [x-www-form-urlencoded](#) [raw](#) [Text](#)

```
1 {  
2   "id": "11",  
3   "password": "4"  
4 }
```

[Send](#) [Save](#) [Preview](#) [Add to collection](#) [Reset](#)

Body [Headers \(4\)](#) **STATUS** 401 Unauthorized **TIME** 62 ms

[Pretty](#) [Raw](#) [Preview](#) [JSON](#) [XML](#)

```
1 User with given id does not exist 11  
2
```

Rysunek 5-4 Odpowiedź serwera w przypadku podania id dla nieistniejącego użytkownika

Put new Data

http://192.168.0.10:8080/Webservice/data PUT URL params Headers (0)

form-data x-www-form-urlencoded raw Text

```

1 {
2   "attentions" : [{"user": 13, "value" : 761, "date" :1384718377776 }],
3   "meditations" : [{"user": 1, "value" : 761, "date" :1384718377776 }],
4   "blinks" : [{"user": 1, "value" : 761, "date" :1384718377776 }],
5   "powers" : [{"user": 1, "value" : 761, "date" :1384718377776 }],
6   "signals" : [{"user": 1, "value" : 761, "date" :1384718377776 }],
7 }

```

Send Save Preview Add to collection Reset

Body Headers (5) STATUS 500 Internal Server Error TIME 78 ms

Pretty Raw Preview JSON XML

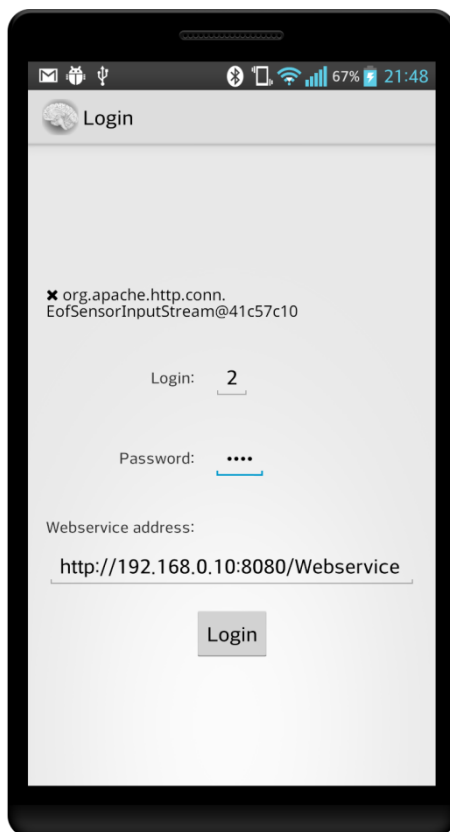
```

1 Cannot add or update a child row: a foreign key constraint fails (`application`.`attentions`,
2 CONSTRAINT `fk_attentions_user` FOREIGN KEY (`userID`) REFERENCES `users` (`id`) ON DELETE NO
  ACTION ON UPDATE NO ACTION)

```

Rysunek 5-5 Podanie niepoprawnego id użytkownika (nieistniejącego) powoduje odrzucenie zapytania

5.4 Uruchamianie aplikacji pracujących wspólnie



Rysunek 5-6 Manifestacja błędu odkrytego w fazie uruchamiania współpracujących aplikacji

Jasno zdefiniowany interfejs komunikacji pomiędzy obiema aplikacjami oraz dokładne testowanie każdej z aplikacji osobno sprawiły, że proces uruchomienia aplikacji pracujących wspólnie, przebiegł bez większych trudności.

Aplikacje pracujące razem testowano nie tylko pod kątem pozytywnych wyników operacji, ale również przewidzianych stanów niepoprawnych. Starano się też wymusić nieoczekiwane sytuacje wyjątkowe.

Łatwym do zaobserwowania i nietrudnym w naprawieniu był błąd związany z odczytem ze strumienia wejściowego, w celu odzyskania ciała nagłówka przy komunikacji z serwerem. Sposób manifestacji błędu został przedstawiony na rysunku obok.

6 Specyfikacja zewnętrzna

6.1 Aplikacja mobilna

6.1.1 Instalacja

Jako że aplikacja nie jest dostępna w serwisie Google Play instalacja aplikacji możliwa jest na dwa sposoby: za pośrednictwem linii poleceń lub z użyciem programu zarządzającego plikami na telefonie.

6.1.1.1 Nieznane źródła

Kroki zezwolenia na instalację aplikacji z nieznanymi źródłami przedstawiono dla telefonu LG P880 z systemem w wersji 4.1.2 i dla innych urządzeń mogą nieznacznie się różnić.

Aby zezwolić na instalację aplikacji z nieznanymi źródłami należy wejść w ustawienia systemu, wybrać kategorię zabezpieczenia (Rysunek 6-1) oraz zaznaczyć pole „Nieznane źródła” (Rysunek 6-2).

6.1.1.2 Z linii poleceń

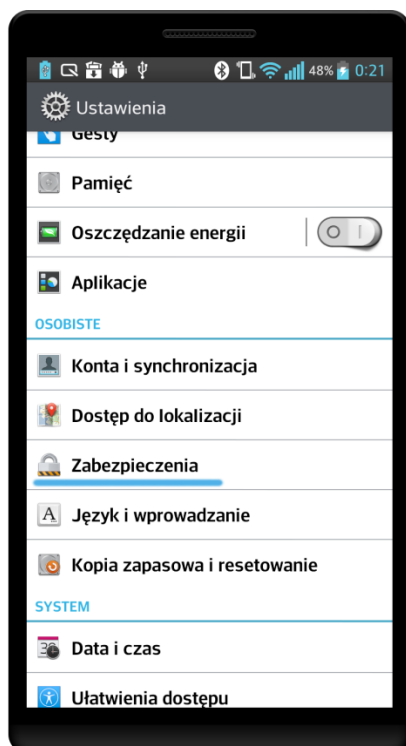
Podane instrukcje dotyczą systemu Windows i zakładają podpięcie jednego urządzenia fizycznego do systemu.

Do przeprowadzenia instalacji tym sposobem wymagane jest posiadanie skonfigurowanego środowiska programistycznego Android. Autor zakłada również, że folder `platform-tools` z folderu SDK dodany jest do zmiennej środowiskowej `PATH`, umożliwiając tym samym wykonanie zawartych w nim programów z dowolnego miejsca w strukturze katalogów.

Należy najpierw wykonać kroki opisane w rozdziale 6.1.1.1, następnie otworzyć konsolę w folderze w którym znajduje się plik `.apk` z aplikacją i wydać polecenie:

```
adb -d install AndroidBachelorThesis.apk
```

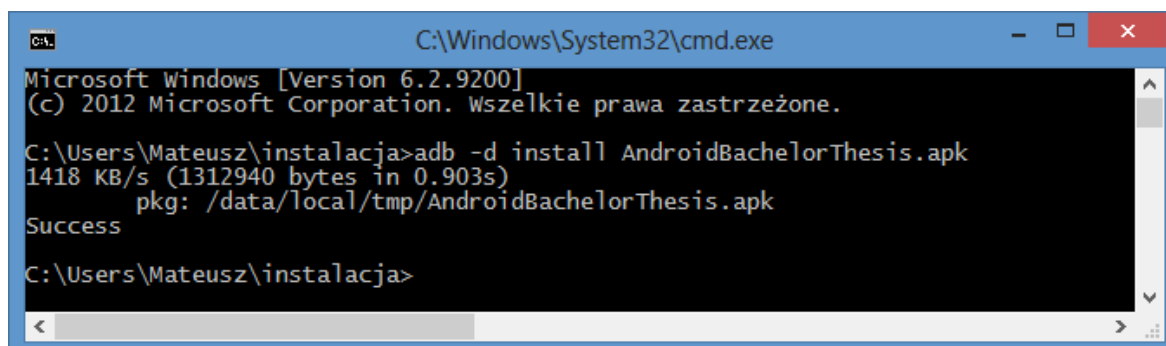
Poprawna instalacja potwierdzona zostanie komunikatem “Success”, tak jak na Rysunek 6-3.



Rysunek 6-1 Widok ustawień systemu



Rysunek 6-2 Opcja, którą należy zaznaczyć aby móc zainstalować aplikację



Rysunek 6-3 Polecenie instalacji aplikacji na urządzeniu

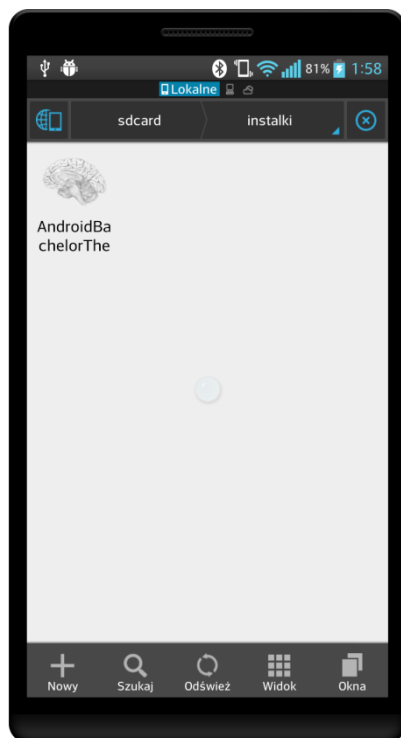
6.1.1.3 Poprzez eksplorator plików

Do instalacji z użyciem eksploratora plików systemu Android również należy najpierw wykonać kroki z rozdziału 6.1.1.1.

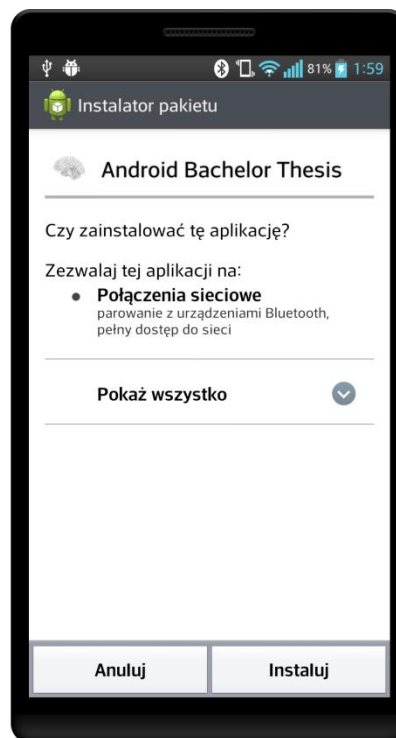
Kolejnym krokiem jest skopiowanie do pamięci telefonu lub karty SD pliku .apk z aplikacją, następnie uruchomienie eksploratora plików (w tym przypadku program ES File Explorer²³) i przejście do folderu, w którym zapisano plik (Rysunek 6-4 Widok eksploratora plików ES File Explorer). Należy teraz dotknąć ikonę pliku i zezwolić na

²³ Adres pod którym program jest dostępny do pobrania:
<https://play.google.com/store/apps/details?id=com.estrongs.android.pop&hl=pl>

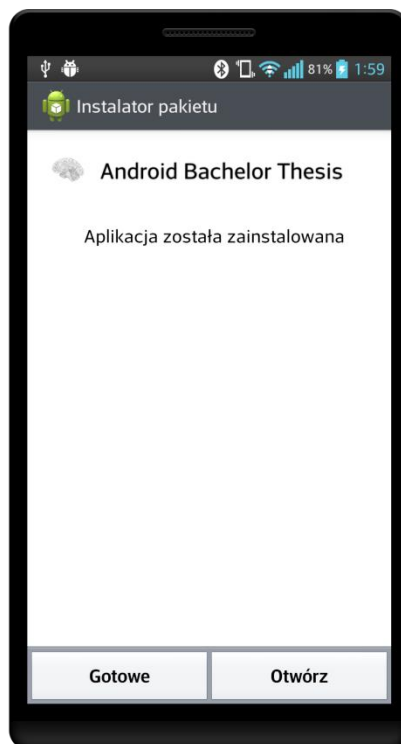
instalację (Rysunek 6-3). Po poprawnej instalacji zostanie wyświetlony ekran potwierdzający prawidłowe zainstalowanie programu (Rysunek 6-6).



Rysunek 6-4 Widok eksploratora plików ES File Explorer



Rysunek 6-5 Monit z potwierdzeniem instalacji

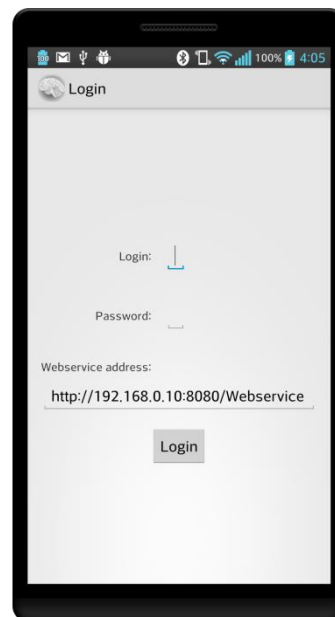


Rysunek 6-6 Potwierdzenie pomyślnie przeprowadzonej instalacji

6.1.2 Użycie

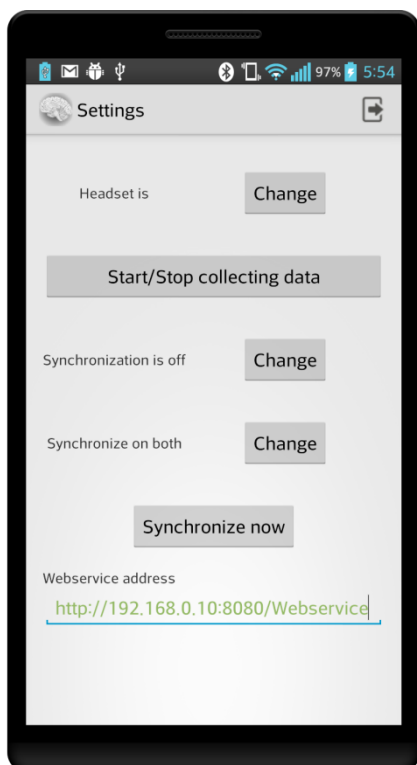
Pierwszą rzeczą jaką musi wykonać użytkownik jest zalogowanie się do aplikacji. Ta funkcja realizowana jest na ekranie logowania widocznym na rysunku 6-7. Użytkownik podaje tutaj dane uzyskane od administratora.

Po poprawnym zalogowaniu użytkownikowi wyświetlony jest główny ekran aplikacji (rysunek 6-8). Ekran ten udostępnia możliwość sterowania momentem synchronizacji danych z serwerem i ustawiania adresu serwera. Do funkcji dotyczących zarządzania zestawem EEG dostępnych z tego poziomu należy zaliczyć: ustanawianie połączenia z obsługą nieaktywnego modułu Bluetooth w urządzeniu oraz sterowanie przesyłaniem danych z zestawu.



Rysunek 6-7 Ekran logowania

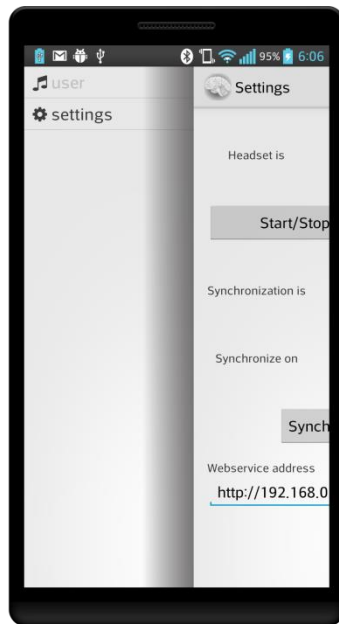
Aby rozpocząć akwizycję danych z czujnika użytkownik powinien najpierw kliknąć przycisk „Change” usytuowany koło pola „Headset is” i odczekać do momentu, aż wspomniane pole zmieni swoją zawartość na „Headset is connected”. Można wtedy zainicjować akwizycję danych przyciskiem znajdującym się bezpośrednio pod polem. Zatrzymania pobierania danych dokonuje się tym samym przyciskiem i powoduje to zmianę wartości pola na „Headset is idle”.



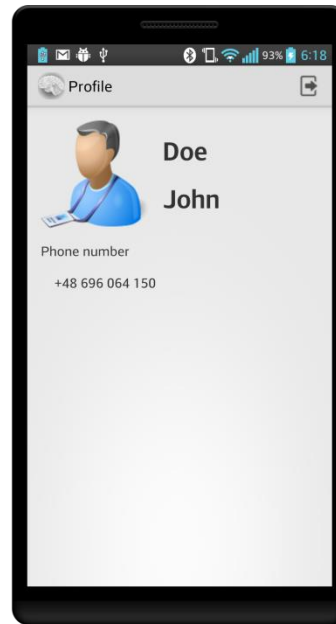
Rysunek 6-8 Główny ekran aplikacji

Użytkownik po wykonaniu ruchu przeciągnięcia (ang. *swype*) od strony lewej do prawej uzyskuje dostęp do menu aplikacji. Służy ono do nawigacji pomiędzy dostępnymi ekranami i przedstawione jest na rysunku 6-9.

Ostatni ekran zawierający informacje o profilu użytkownika jest przedstawiony na rysunku 6-10. Nie ma on żadnej wartości funkcjonalnej, jedynie informacyjną. Należy też wspomnieć o tym, że numer telefonu i zdjęcie są w tym momencie stałe.



Rysunek 6-10 Menu aplikacji



Rysunek 6-9 Ekran profilu

6.2 Aplikacja serwerowa

6.2.1 Instalacja

6.2.1.1 MySQL

Aplikacja serwerowa do poprawnego działania wymaga zewnętrznej bazy MySQL, jej instalator można pobrać z podstrony projektu²⁴. Autor zakłada, że folder zawierający²⁵ program `mysql` został dodany do zmiennej środowiskowej `PATH`.

Używając programu `mysql` dostarczanego razem z bazą danych trzeba wykonać skrypt tworzący bazę danych. Poniższe polecenie zakłada, że skrypt nazywa się `createStatements.txt` i wywołanie polecenia następuje z katalogu bezpośrednio go zawierającego.

```
mysql < createStatements.txt -user=<nazwa_użytkownika>
--password=<hasło_użytkownika>
```

Należy przy tym pamiętać, że podany użytkownik musi mieć nadane odpowiednie prawa.

²⁴ adres <http://dev.mysql.com/downloads/installer/>

²⁵ Jest to folder MySQL Workbench CE z katalogu instalacyjnego.

6.2.1.2 Glassfish

Po pobraniu serwera Glassfish należy go rozpakować i dodać folder bin z katalogu do zmiennej PATH. Następnie skopiować załączoną do pracy bibliotekę MySQL Connector²⁶ do katalogu <ścieżka_instalacji_serwera>/glassfish/lib.

Teraz należy przejść do katalogu, który zawiera plik .war z aplikacją i wywołać następujące komendy:

```
asadmin start-domain domain1
asadmin deploy Webservice.war
```

Jeśli polecenia zostały wykonane bez błędów, oznacza to, że wszystko jest skonfigurowane poprawnie.

6.2.2 Użycie

Używanie aplikacji serwerowej sprowadza się do dwóch czynności. Wyświetlania danych pobranych od poszczególnych użytkowników oraz dodawania użytkowników.

Aby wykonać obie wymienione czynności należy rozpocząć od odwiedzenia strony /view.jsp. Na tej stronie jest wystawiany punkt logowania administratora (rysunek 6-11). Domyślne dane do logowania to użytkownik admin z hasłem concept.



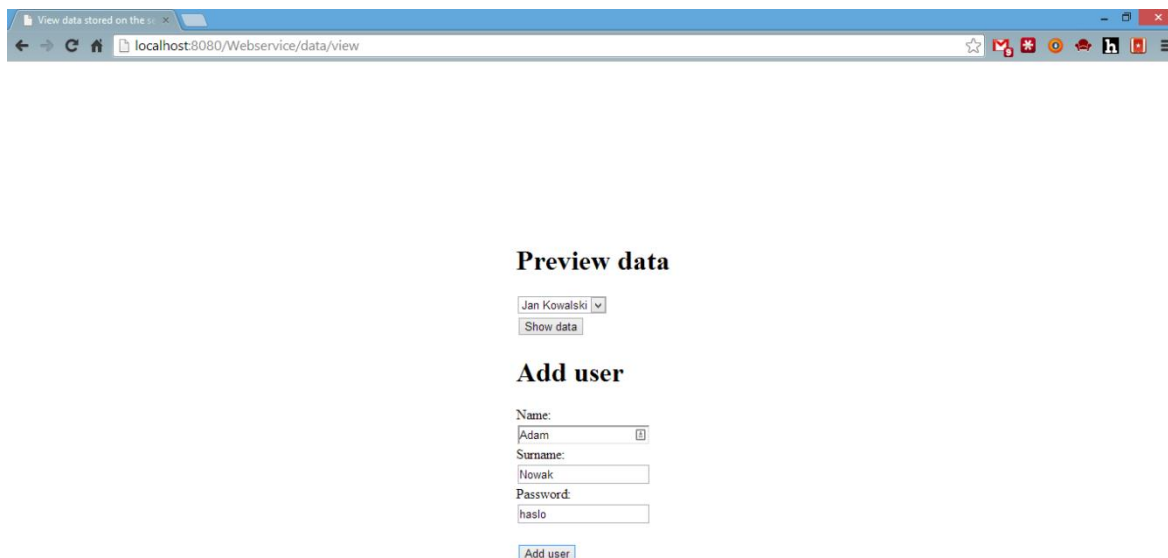
Rysunek 6-11 Punkt logowania administratora

Po zalogowaniu dostępne są dwie akcje: wyświetlanie danych dla danego użytkownika oraz dodawanie użytkowników do bazy danych (rysunek 6-12). W celu wyświetlenia pobranych od użytkowników danych, należy wybrać użytkownika z rozwijanej listy, a następnie kliknąć przycisk „Show data”. W efekcie wyświetlona zostanie lista danych dla konkretnego użytkownika, podobna do tej z rysunku 6-13.

Aby dodać użytkownika na stronie widocznej zaraz po zalogowaniu, należy wypełnić wszystkie pola, a następnie kliknąć przycisk „Add user”. Spowoduje to przeładowanie strony i, w zależności od poprawności podanych danych (pola nie mogą być puste), pokazanie odpowiedniego komunikatu. Przykładowy komunikat wyświetlony po

²⁶ Można ją znaleźć na załączonym do pracy dysku w folderze Pliki instalacyjne > Aplikacja serwerowa lub pobrać z Internetu z adresu <http://dev.mysql.com/downloads/connector/j/>

poprawnym dodaniu użytkownika jest widoczny na rysunku 6-14, a po podaniu niepoprawnych danych na rysunku 6-15.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Webservice/data/view'. The page content is divided into two main sections:

Preview data

Below this heading, there is a dropdown menu currently showing 'Jan Kowalski' and a 'Show data' button.

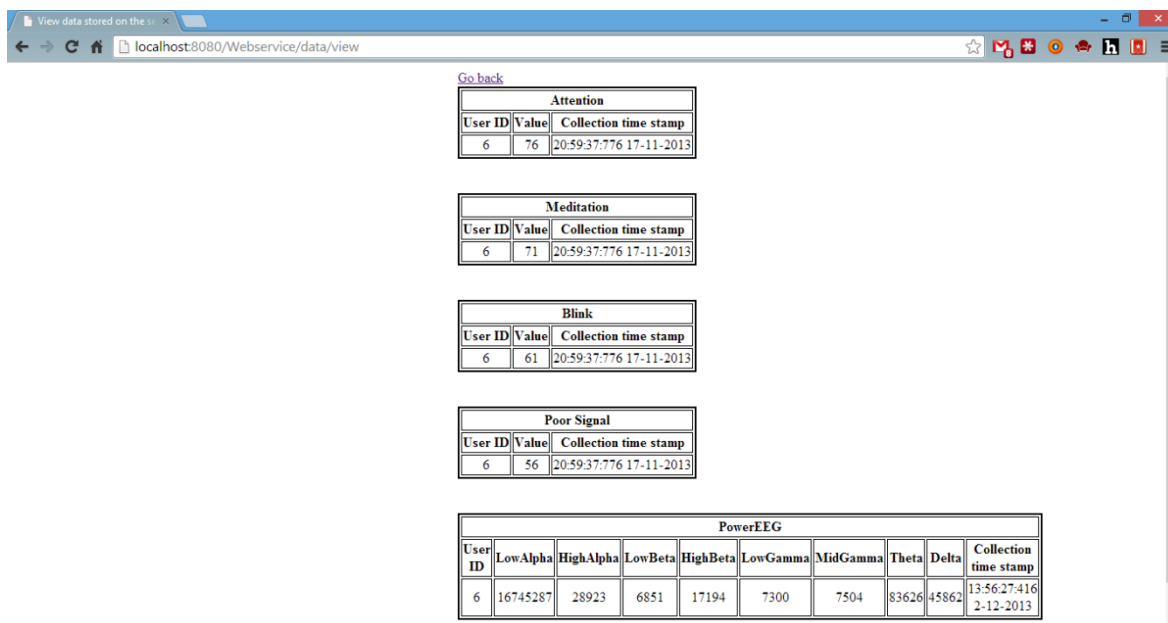
Add user

Below this heading, there is a form with the following fields:

- Name: (text input, value: Adam)
- Surname: (text input, value: Nowak)
- Password: (text input, value: haslo)

At the bottom of the form is an 'Add user' button.

Rysunek 6-12 Dostępne akcje



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Webservice/data/view'. The page content includes a 'Go back' link and five data tables for User ID 6:

Attention

User ID	Value	Collection time stamp
6	76	20:59:37:776 17-11-2013

Meditation

User ID	Value	Collection time stamp
6	71	20:59:37:776 17-11-2013

Blink

User ID	Value	Collection time stamp
6	61	20:59:37:776 17-11-2013

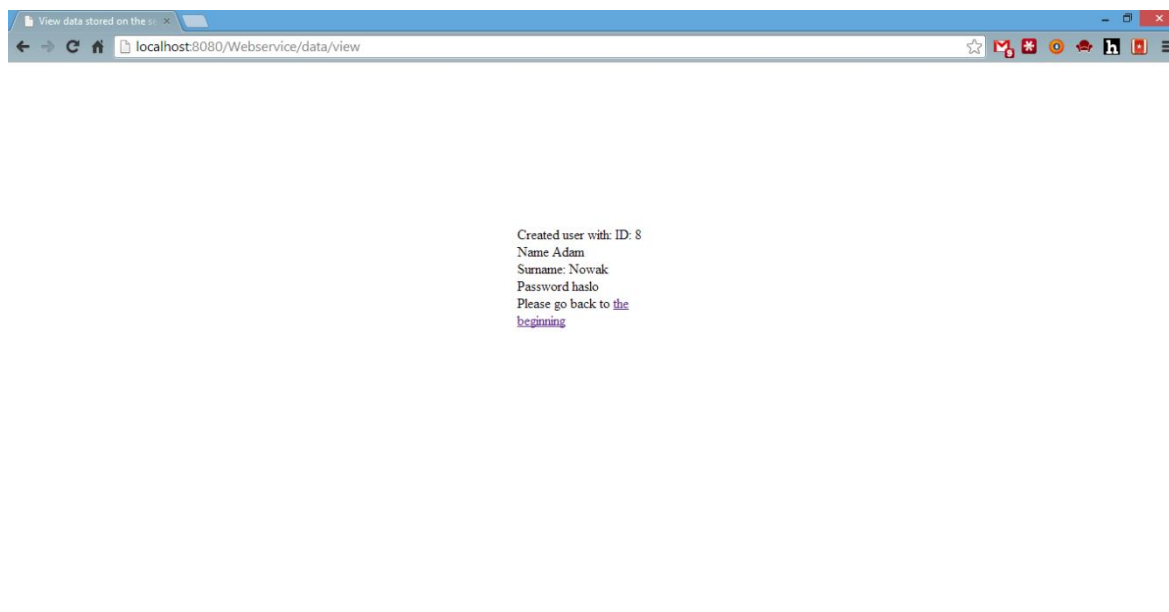
Poor Signal

User ID	Value	Collection time stamp
6	56	20:59:37:776 17-11-2013

PowerEEG

User ID	LowAlpha	HighAlpha	LowBeta	HighBeta	LowGamma	MidGamma	Theta	Delta	Collection time stamp
6	16745287	28923	6851	17194	7300	7504	83626	45862	13:56:27:416 2-12-2013

Rysunek 6-13 Lista danych użytkownika, tutaj zawierająca tylko po jednej wartości dla każdego ze wskaźników



Rysunek 6-14 Komunikat potwierdzający poprawne dodanie użytkownika



Rysunek 6-15 Komunikat informujący o niepoprawnie wypełnionym formularzy rejestracyjnym

7 Uwagi końcowe

W ramach projektu dyplomowego wykonano projekt oraz implementację systemu pozwalającego monitorować aktywność elektryczną mózgu bez potrzeby wizyty w placówce oraz w sposób w miarę nieinwazyjny (rozpatrując wygodę użytkownika).

Projekt i wykonanie dwóch aplikacji, które muszą komunikować się ze sobą a tworzone są niejednocześnie, nie pozwalając tym samym na testowanie komunikacji na bieżąco pozwoliły Autorowi docenić wagę fazy projektowania systemu oraz podążania za ustalonymi wytycznymi w całym procesie wytwarzania oprogramowania.

Przeprowadzenie projektu pozwoliło Autorowi utwierdzić się w wartości, jaką niesie ze sobą testowanie automatyczne, niestety, w samej pracy brak jest takich testów. Są one jednak pierwszym celem w przypadku zdecydowania się na rozszerzanie funkcjonalności aplikacji.

Wybór niedojrzałego i tym samym niedopracowanego jeszcze środowiska IDE, jakim na początku pisania aplikacji było Android Studio powodowało większość kłopotów napotkanych podczas pisania pracy. Tworzyło to nieprzyjemny efekt walki z narzędziem, a nie zadaniem. Jednak w miarę udostępniania przez twórców kolejnych poprawek, to środowisko przestawało być problematyczne i zaczynało pomagać, a nie przeszkadzać. Rodzi to jednak pytanie na przyszłość czy na pewno warto wybierać wchodzące produkty do takich zadań.

Z części implementacyjnej warto wspomnieć o mechanizmie przekazywania obiektów przez bariery procesów i związane z nim niedogodności, które początkowo spowodowały spowolnienie pracy, jednak dość szybko zostały rozwiązane.

Wszystkie wyznaczone dla systemu wymagania zostały spełnione. Łatwe spełnienie założeń (względem aplikacji mobilnej) było podyktowane nie tylko udostępnianymi przez system możliwościami (dostępem do stanu sieci etc.), ale także odpowiednim przemyśleniem projektu we wczesnych fazach.

Zrealizowany projekt w założeniu jest podstawą, wokół której dobudowane mogą być kolejne funkcjonalności. Przyszłych, możliwych rozszerzeń lub ponownych uzyć jest bardzo dużo, aby wymienić kilka z nich, można wspomnieć o osobistym asystencie kontaktu z przychodnią, który, nie wymagając większej wiedzy (poza odpowiadającymi terminami) i uwagi od użytkownika, mógłby ustalać konsultacje u specjalisty (po zebraniu odpowiedniej ilości danych).

Projektowi można też nadać nowe życie, wykorzystując go jako mechanizm sterowania grami wieloosobowymi, zapewniający warstwę przesyłu danych. Dodatkowym atutem tego podejścia jest brak większych modyfikacji, poza dostosowaniem do używanego protokołu (w rozumieniu schematu przesyłanych do serwera danych).

Jak widać, zrealizowany projekt jest na tyle ogólny i elastyczny, że ponowne użycie go, jako część kolejnych projektów, wydaje się nie tylko wskazane ale konieczne.

8 Bibliografia

- [1] Fielding T. R., Taylor N. R., (Maj 2002) *Principled Design of the Modern Web Architecture*, ACM Transactions on Internet Technology, Vol. 2, No. 2, , Pages 115–150 <http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf> [25 grudzień 2013]
- [2] Larsen A. E., (Czerwiec 2011) *Classification of EEG Signals in a Brain-Computer Interface System*
- [3] Zhang, Y., Chen, Y., Bressler, S. L., & Ding, M. (2009). *Response preparation and inhibition: The role of the cortical sensorimotor beta rhythm*. Neuroscience, 156:1, 238–246. <http://www.ccs.fau.edu/~bressler/pdf/Neuroscience08.pdf> [29 grudzień 2013]
- [4] Collins, C., Galpin, M., Kaeppler, M., (2012) *Android w praktyce* tłumaczył Walczak, T., Helion, Gliwice
- [5] Gamma, E., Helm, R., Johnson, R., Vlissides, J., (2010) *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* tłumaczył Walczak, T., Helion, Gliwice
- [6] Fowler M., (2004) *Inversion of Control Containers and the Dependency Injection pattern*, <http://www.martinfowler.com/articles/injection.html> [10-01-2014]

9 Spis rysunków

Rysunek 2-1 Koncepcja wzorca model widok kontroler, źródło	2
Rysunek 2-2 Zestawienie głównych klas fal mózgowych, źródło [2], strona 11	4
Rysunek 2-3 Rozmieszczenie elektrod w systemie 10-20, źródło	4
Rysunek 2-4 Architektura systemu, źródło własne.	5
Rysunek 3-1 Podział aplikacji mobilnej na procesy.....	7
Rysunek 3-2 Struktura klas w procesie akwizycji danych	8
Rysunek 3-3 Struktura klas uczestniczących w autoryzacji użytkownika w procesie danych	9
Rysunek 3-4 Diagram sekwencji pokazujący realizację zadania autoryzacji użytkownika wewnątrz procesu danych.....	10
Rysunek 3-5 Struktura klas części procesu odpowiedzialnego za obsługę danych	10
Rysunek 3-6 Klasy biorące udział w synchronizacji danych	11
Rysunek 3-7 Klasy wykonujące komunikację z procesem akwizycji	12
Rysunek 3-8 Łącznik udostępniający funkcję autoryzacji	13
Rysunek 3-9 Klasy łącznika danych.....	14
Rysunek 3-10 Klasy łącznika komend	15
Rysunek 3-11 Schemat bazy danych	19
Rysunek 4-1 Zasada działania biblioteki greenDAO, źródło strona projektu	21
Rysunek 5-1 Wizualizacja modelu V, źródło	29
Rysunek 5-2 Zrzut ekranu pokazujący program użyty do testowania części serwerowej ..	30
Rysunek 5-3 Odpowiedź serwera w przypadku braku autoryzacji (podane hasło jest niepoprawne)	31
Rysunek 5-4 Odpowiedź serwera w przypadku podania id dla nieistniejącego użytkownika	31
Rysunek 5-5 Podanie niepoprawnego id użytkownika (nieistniejącego) powoduje odrzucenie zapytania	32
Rysunek 5-6 Manifestacja błędu odkrytego w fazie uruchamiania współpracujących aplikacji	32
Rysunek 6-1 Widok ustawień systemu.....	34
Rysunek 6-2 Opcja, którą należy zaznaczyć aby móc zainstalować aplikację	34
Rysunek 6-3 Polecenie instalacji aplikacji na urządzeniu.....	34
Rysunek 6-4 Widok eksploratora plików ES File Explorer	35

Rysunek 6-5 Monit z potwierdzeniem instalacji	35
Rysunek 6-6 Potwierdzenie pomyślnie przeprowadzonej instalacji	35
Rysunek 6-7 Ekran logowania	36
Rysunek 6-8 Główny ekran aplikacji	36
Rysunek 6-9 Ekran profilu	37
Rysunek 6-10 Menu aplikacji	37
Rysunek 6-11 Punkt logowania administratora	38
Rysunek 6-12 Dostępne akcje	39
Rysunek 6-13 Lista danych użytkownika, tutaj zawierająca tylko po jednej wartości dla każdego ze wskaźników	39
Rysunek 6-14 Komunikat potwierdzający poprawne dodanie użytkownika	40
Rysunek 6-15 Komunikat informujący o niepoprawnie wypełnionym formularzy rejestracyjnym	40

10 Spis fragmentów kodu

Listing 3-1 Format zapytania weryfikującego tożsamość użytkownika	16
Listing 3-2 Format odpowiedzi zawierającej ostatnie czasy aktualizacji	17
Listing 3-3 Format danych przesyłanych do serwera w celu utrwalenia	17
Listing 3-4 Obiekt JSON reprezentujący wartości prostego wskaźnika	17
Listing 3-5 Obiekt JSON reprezentujący wartości złożonego wskaźnika	18
Listing 4-1 Nagłówek i pola klasy MainService	23
Listing 4-2 Wybrane fragmenty klasy EEGAcquisitionService	25
Listing 4-3 Metoda udostępniająca połączenie z bazą danych	27
Listing 4-4 Fragment klasy dokonującej odczytu z bazy danych	27
Listing 4-5 Fragment klasy zapisującej dane do bazy danych	28

Dodatek A Zawartość CD

Struktura katalogów:

- Praca dyplomowa
 - Zawiera pracę dyplomową w formacie `.pdf` oraz `.docx`
- Pliki instalacyjne
 - Aplikacja mobilna
 - Zawiera plik z rozszerzeniem `.apk` z aplikacją na system Android
 - Aplikacja serwerowa
 - Folder zawiera plik `.war` do bezpośredniego rozpakowania na serwerze oraz bibliotekę niezbędną do uruchomienia.
- Kod źródłowy
 - Aplikacja mobilna
 - Kompletny katalog projektu dla środowiska Android Studio w wersji 0.3.7 zawierający kod źródłowy aplikacji mobilnej
 - Aplikacja serwerowa
 - Kompletny katalog projektu dla środowiska Eclipse w wersji Kepler zawierający kod źródłowy aplikacji serwerowej