

**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](#_Toc377579895)

[2 Analiza tematu 2](#_Toc377579896)

[2.1.1 Pojęcia ogólne 2](#_Toc377579897)

[2.2 EEG, fale mózgowe 3](#_Toc377579898)

[2.2.1 Alfa 3](#_Toc377579899)

[2.2.2 Beta 3](#_Toc377579900)

[2.2.3 Delta 3](#_Toc377579901)

[2.2.4 Gamma 3](#_Toc377579902)

[2.2.5 Theta 3](#_Toc377579903)

[2.3 EMG, napięcie mięśniowe 5](#_Toc377579904)

[2.4 Wymagania stawiane aplikacji 5](#_Toc377579905)

[2.5 Wykorzystywane narzędzia 5](#_Toc377579906)

[3 Organizacja wewnętrzna 6](#_Toc377579907)

[3.1 Aplikacja mobilna 6](#_Toc377579908)

[3.1.1 Ogólny opis 6](#_Toc377579909)

[3.1.2 Struktura i działanie procesu akwizycji 7](#_Toc377579910)

[3.1.3 Struktura i działanie procesu danych 9](#_Toc377579911)

[3.1.4 Komunikacja międzyprocesowa 12](#_Toc377579912)

[3.1.5 Organizacja bazy danych 15](#_Toc377579913)

[3.2 Aplikacja serwerowa 16](#_Toc377579914)

[3.2.1 Ogólny opis 16](#_Toc377579915)

[3.2.2 Udostępniane punkty końcowe 16](#_Toc377579916)

[3.2.2.1 Logowanie 16](#_Toc377579917)

[3.2.2.2 Synchronizacja różnicowa 16](#_Toc377579918)

[3.2.2.3 Zapisywanie danych 17](#_Toc377579919)

[3.2.2.4 Weryfikacja serwisu 18](#_Toc377579920)

[3.2.2.5 Podgląd danych 18](#_Toc377579921)

[3.2.3 Organizacja bazy danych 19](#_Toc377579922)

[4 Implementacja 20](#_Toc377579923)

[4.1 Aplikacja mobilna 20](#_Toc377579924)

[4.1.1 Użyte biblioteki 20](#_Toc377579925)

[4.1.2 Szczegóły implementacji 21](#_Toc377579926)

[4.2 Aplikacja serwerowa 25](#_Toc377579927)

[4.2.1 Użyte biblioteki 25](#_Toc377579928)

[4.2.2 Szczegóły implementacji 25](#_Toc377579929)

[5 Testowanie i uruchamianie 28](#_Toc377579930)

[5.1 Model wzorcowy 28](#_Toc377579931)

[5.2 Testowanie aplikacji mobilnej 29](#_Toc377579932)

[5.3 Testowanie aplikacji serwerowej 29](#_Toc377579933)

[5.4 Uruchamianie aplikacji pracujących wspólnie 31](#_Toc377579934)

[6 Specyfikacja zewnętrzna 32](#_Toc377579935)

[6.1 Aplikacja mobilna 32](#_Toc377579936)

[6.1.1 Instalacja 32](#_Toc377579937)

[6.1.1.1 Nieznane źródła 32](#_Toc377579938)

[6.1.1.2 Z linii poleceń 32](#_Toc377579939)

[6.1.1.3 Poprzez eksplorator plików 33](#_Toc377579940)

[6.1.2 Użycie 35](#_Toc377579941)

[6.2 Aplikacja serwerowa 36](#_Toc377579942)

[6.2.1 Instalacja 36](#_Toc377579943)

[6.2.1.1 MySQL 36](#_Toc377579944)

[6.2.1.2 Glassfish 37](#_Toc377579945)

[6.2.2 Użycie 37](#_Toc377579946)

[7 Uwagi końcowe 40](#_Toc377579947)

[8 Bibliografia 42](#_Toc377579948)

[9 Spis rysunków 43](#_Toc377579949)

[10 Spis fragmentów kodu 44](#_Toc377579950)

# Wstęp

Ciągle postępująca miniaturyzacja elementów elektronicznych, której towarzyszy jednoczesne 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 osadzane 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.

# Analiza tematu

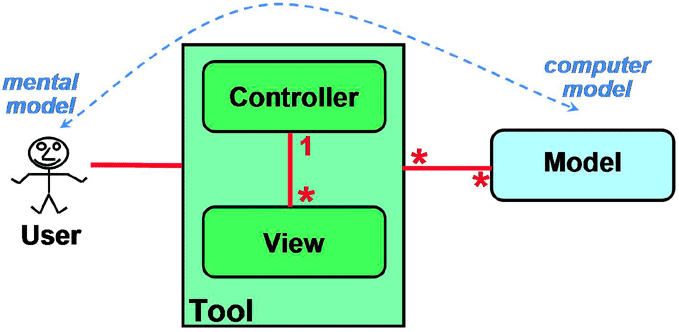
W celu pełnego zrozumienia omawianych zagadnień, w tym rozdziale przytoczono definicje używanych w pracy pojęć i narzędzi.

### Pojęcia ogólne

Poniżej przytoczono definicję pochodzącą ze strony[[1]](#footnote-1) 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 ‑ Koncepcja wzorca model widok kontroler, źródło[[2]](#footnote-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 umiejscawiany 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.

### 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).

### Beta

Fale z zakresu 12 do 30 Hz, często wyróżnia się fale β1 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.

### Delta

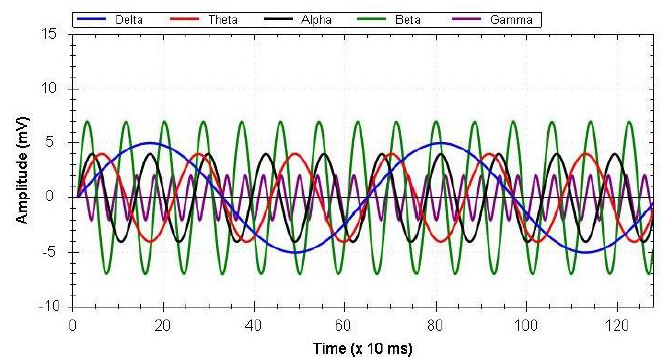
Zakres tych fal sięga od 0,5 do 3,5 Hz. Są najwolniejszymi z fal, u normalnie rozwiniętych, dorosłych osób występują jedynie podczas snu. Obecność tych fal  
u przytomnych osób wskazuje defekt mózgu.

### Gamma

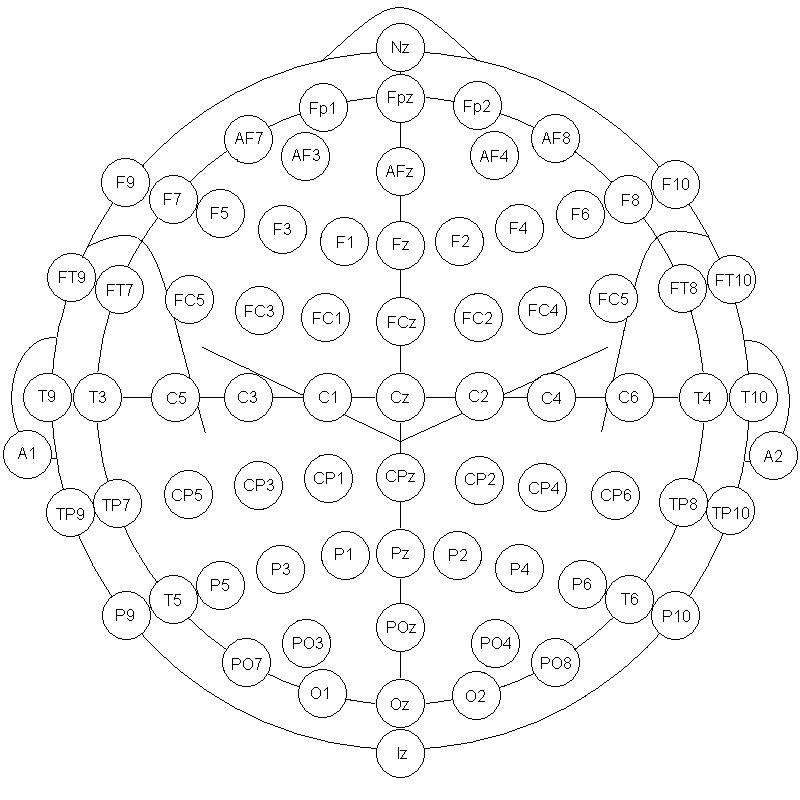
Fale z zakresu częstotliwości większego niż 31 Hz. Razem z falami beta powiązane są ze skupieniem, mechanizmami poznawania i rozpoznawania.

### 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 ‑ Zestawienie głównych klas fal mózgowych, źródło [2], strona 11



Rysunek ‑ Rozmieszczenie elektrod w systemie 10-20, źródło[[3]](#footnote-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.

## Wymagania stawiane aplikacji

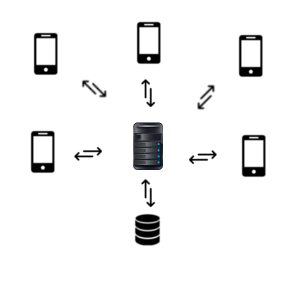
Założenia funkcjonalne stawiane aplikacjom wyglądają następująco:

* obsługa kont użytkownika (logowanie),
* różnicowa synchronizacja danych.

Założenia niefunkcjonalne:

* zgodność z systemem Android >= 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 ‑ Architektura systemu, źródło własne.

## 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 projektu[[4]](#footnote-4) oraz serwera Glassfish w wersji 4.0, którą również pobrano ze strony projektu[[5]](#footnote-5). Część mobilną napisano z użyciem środowiska dostarczanego przez Google i dedykowanego do pisania aplikacji na system   
Android – Android Studio, który można pobrać z Internetu[[6]](#footnote-6). 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 Android[[7]](#footnote-7).

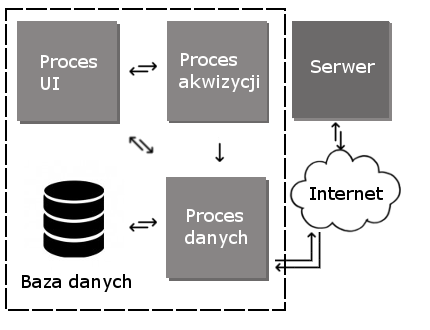
# Organizacja wewnętrzna

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

## Aplikacja mobilna

### 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 zarządzeniem połączeniem z zestawem Neurosky MindWave Mobile (dalej w Pracy nazywany zestawem) oraz obsługą danych przychodzących z zestawu. Jest on uruchamiany za pośrednictwem obiektu klasy EEGAcquisitionServiceConnectionConnector.

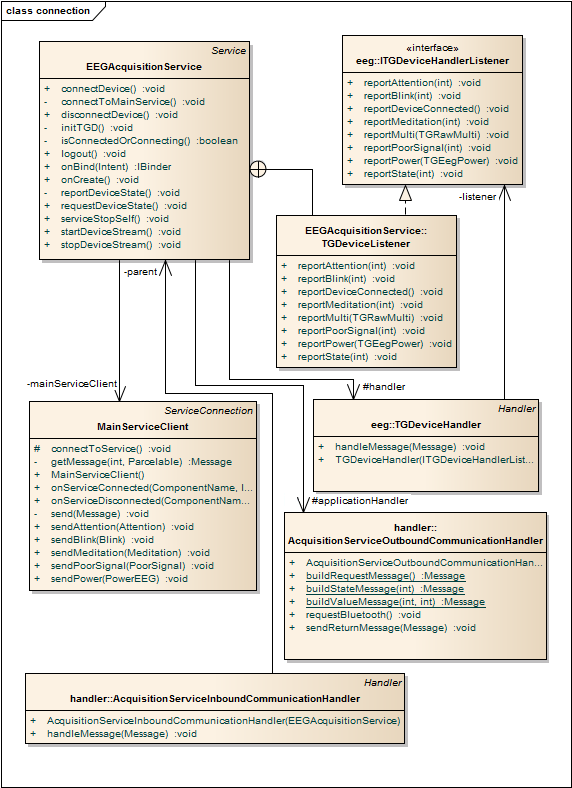


Rysunek ‑ Podział aplikacji mobilnej na procesy

### 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

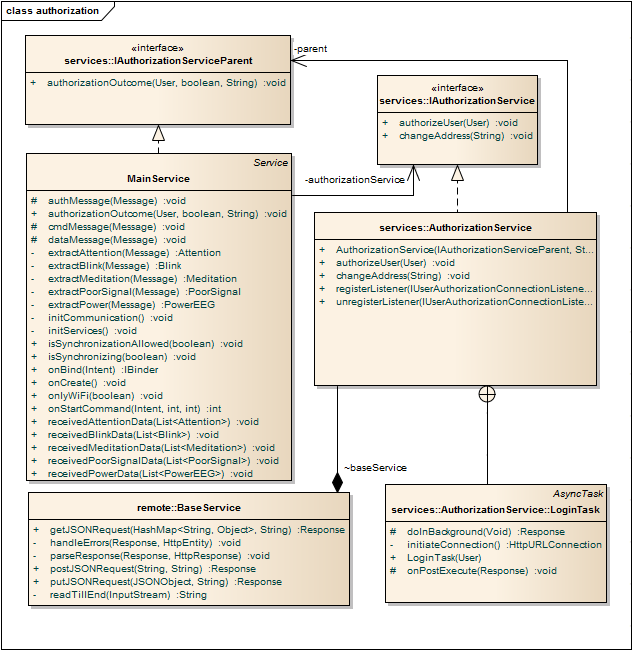
AcquisitionServiceOutboundCommunicationHandler, które zajmują się odpowiednio obsługą przyjmowanych komunikatów oraz tworzeniem i przesyłaniem wiadomości do procesu UI. Obiekty klasy odpowiadającej za przyjmowanie komunikatów dekodują wiadomości zgodnie z przyjętą reprezentacją (wykorzystano tutaj pola 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 ‑ Struktura klas w procesie akwizycji danych

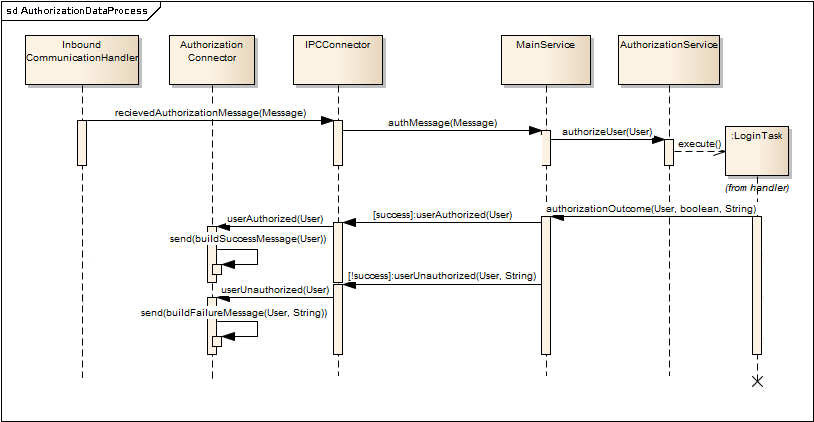
### Struktura i działanie procesu danych

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.

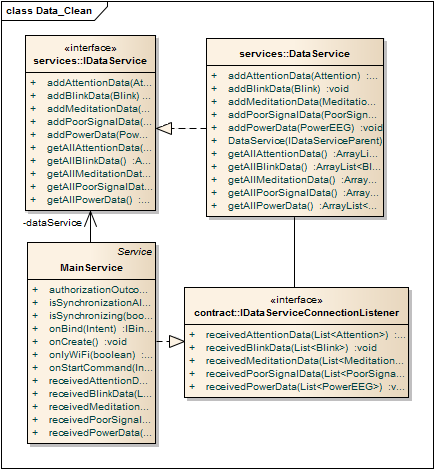


Rysunek ‑ Struktura klas uczestniczących w autoryzacji użytkownika w procesie danych

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



Rysunek ‑ Diagram sekwencji pokazujący realizację zadania autoryzacji użytkownika wewnątrz procesu danych

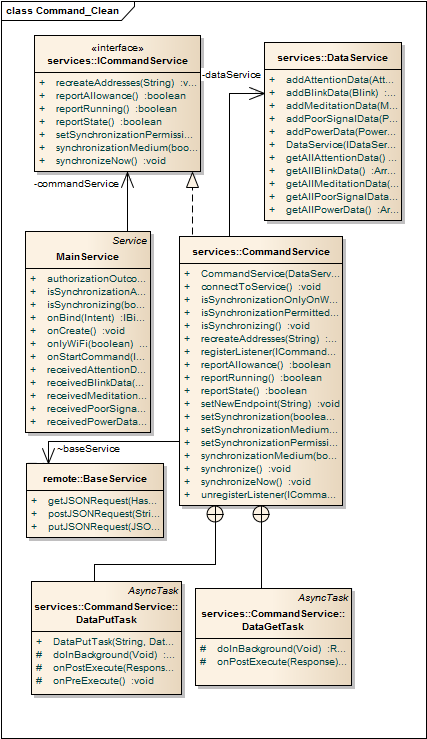
Druga z omawianych części jest odpowiedzialna za obsługę danych. Musi ona umożliwić odczyt i zapis danych oraz odczyt danych od określonej daty. Ostatnia  
z wymienionych funkcjonalności jest konieczna, aby spełnić jedno z kluczowych założeń funkcjonalnych aplikacji, wymaganie synchronizacji różnicowej.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

Rysunek ‑ Struktura klas części procesu odpowiedzialnego za obsługę danych

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.

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

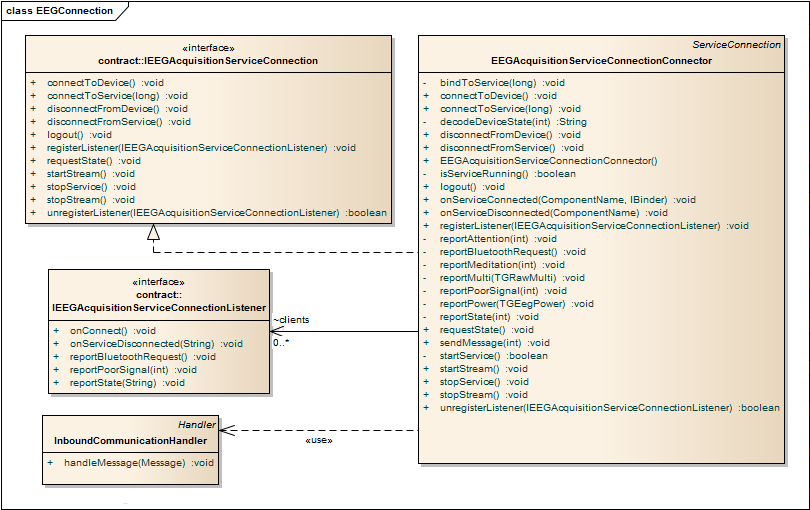
DataService do odzyskania danych z lokalnej bazy danych oraz wcześniej omówionych klas do synchronizacji.

Rysunek ‑ Klasy biorące udział w synchronizacji danych

### 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 serwisów[[8]](#footnote-8) (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.

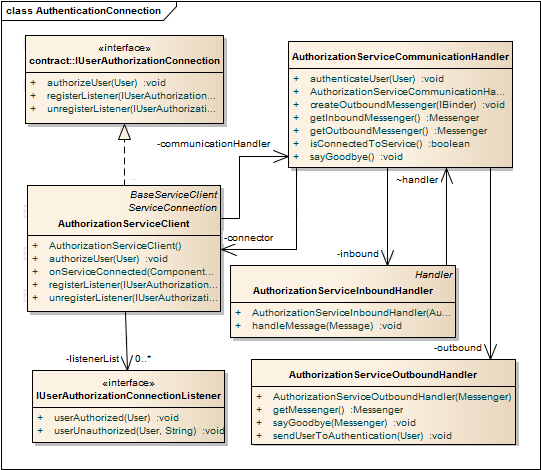


Rysunek ‑ Klasy wykonujące komunikację z procesem akwizycji

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 komunikacją zwrotną. Jednocześnie klasa łącznika (tutaj EEGAcquisitionServiceConnectionConnector) implementuje interfejs, poprzez który klienci łącznika mogą zwracać się do niego w celu wykonania określonego 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 Language[[9]](#footnote-9)) jedyna zmiana, 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](#_Struktura_i_działanie).

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 ‑ Łącznik udostępniający funkcję autoryzacji

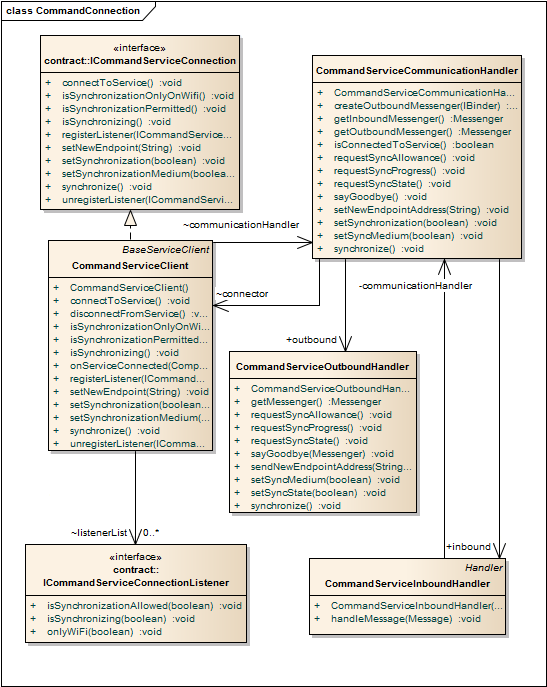
We wszystkich łącznikach komunikujących się z procesem danych wprowadzono dodatkowe klasy, AuthorizationServiceCommunicationHandler,

AuthorizationServiceInboundHandler oraz AuthorizationService OutboundHandler. 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 AuthorizationService InboundHandler.

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



Rysunek ‑ Klasy łącznika danych



Rysunek ‑ Klasy łącznika komend

### 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.

## Aplikacja serwerowa

### 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.

### Udostępniane punkty końcowe

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

#### 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 ‑ 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).

#### 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 ‑ Format odpowiedzi zawierającej ostatnie czasy aktualizacji

#### 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 ‑ 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 ‑ 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 ‑ Obiekt JSON reprezentujący wartości złożonego wskaźnika

Serwlet w odpowiedzi na żądanie zapisu zwraca nagłówek http z odpowiednim kodem statusu (ang. status code) oraz komunikatem przekazanym 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.

#### Weryfikacja serwisu

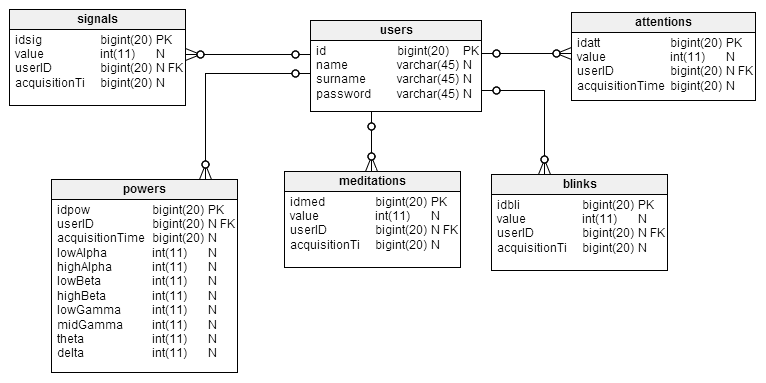
W celu ułatwienia weryfikacji adresu podanego przez użytkownika (wprowadzanego ręcznie w aplikacji mobilnej) przygotowano punkt końcowy, który na każde zapytanie GET lub POST odsyła odpowiedź z nagłówkiem http, którego pole statusu jest ustawiane na wartość 200. Ten punkt końcowy jest obsługiwany przez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

#### 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 odnośnie wykonania jednej z wymienionych funkcji podejmowana jest na podstawie wartości parametru action przekazywanego w nagłówku.

### 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 ‑ Schemat bazy danych

# Implementacja

## Aplikacja mobilna

### Użyte biblioteki

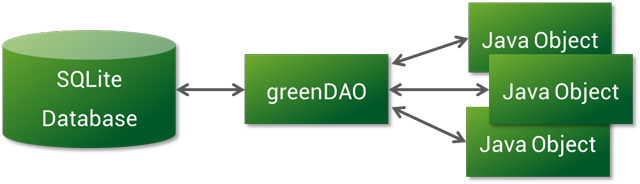
Obecnie budowanie aplikacji bez użycia zewnętrznych bibliotek jest nie tyle co niemożliwe a niepraktyczne i zbędne. Biblioteki dostarczają gotowych rozwiązań najczęściej występujących, powtarzających się problemów, pomagają w rozwiązywaniu już istniejących lub zwyczajnie ułatwiają pracę programiście.

W projekcie zastosowano kilka bibliotek, pierwszą z opisanych będzie stworzony przez Jake’a Whartona Butterknife[[10]](#footnote-10). 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 Dagger[[11]](#footnote-11) stworzony przez firmę Square. Biblioteka ta jest lekkim (ang. lightweight) i szybkim frameworkiem do wstrzykiwania zależności bazującym na adnotacjach javax.inject[[12]](#footnote-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 konfigurujące instancję klasy. Dodatkowo nie wymusza dziedziczenia po jakiejkolwiek klasie, jak ma to miejsce w innych frameworkach (RoboGuice[[13]](#footnote-13)). Jego możliwości są o wiele większe aniżeli opisano w tym tekście, aby dowiedzieć się więcej, warto zapoznać się z nimi na stronie projektu13.

SlidingMenu[[14]](#footnote-14) 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), greenDAO[[15]](#footnote-15). Dzięki niej z programisty zostaje zdjęty obowiązek utrzymywania wszystkich zapytań i konwersji obiektów na encje oraz encji na obiekty.



Rysunek ‑ Zasada działania biblioteki greenDAO, źródło strona projektu15

Ostatnią biblioteką użytą w projekcie jest ThinkGear[[16]](#footnote-16), 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.

### 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 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** **IPCConnector** 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(IPCConnector.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 ‑ 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 żądanego 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 tyko raz dla każdego obiektu.

Konieczną do opisania 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 ‑ Wybrane fragmenty klasy EEGAcquisitionService

## Aplikacja serwerowa

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

### Użyte biblioteki

Jednym z głównych zadań aplikacji serwerowej jest komunikacja z bazą danych. Aby umożliwić realizację tego zadania należało skorzystać z biblioteki MySQL Connector[[17]](#footnote-17) dla języka Java.

Użyto również dwóch bibliotek z projektu Apache Commons[[18]](#footnote-18), 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-java[[19]](#footnote-19) do obsługi przesyłanych wiadomości.

### 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.

**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 ‑ 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 ‑ 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,

OperationResult result) {

5: **try** {

6: persistAttention(dataContainer.getAttentions());

7: persistMeditation(dataContainer.getMeditations());

8: persistBlink(dataContainer.getBlinks());

9: persistPowers(dataContainer.getPowers());

10: persistSignals(dataContainer.getSignals());

11: } **catch** (SQLException e) {

12: result.outcome = e.getMessage();

13: result.code =

HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR;

14: }

15: }

16:

17: **private void** persistAttention(List<Attention> signals)

**throws** SQLException {

18: Connection connection =

DBAccessor.getConnection();

19: Statement createStatement =

connection.createStatement();

20: **for** (Attention att : signals) {

21: String sql = "INSERT INTO `"

+ SCHEMA + "`.`attentions` (value,userID,acquisitionTime)"

+ "VALUES(" + att.getValue() + "," + att.getUserId() + ","

+ att.getCollectionDate().getTime() + ") ";

22: createStatement.addBatch(sql);

23: }

24: createStatement.executeBatch();

25: connection.close();

26: }

Listing ‑ Fragment klasy zapisującej dane do bazy danych

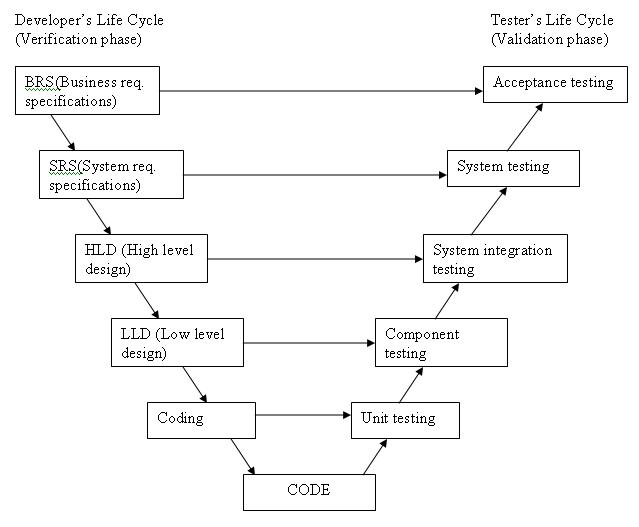
Fragment klasy DBWriter przedstawiony na Listing 4‑5 zawiera główną metodę używaną do zapisu paczki danych otrzymanych od klienta (persistData()).   
Jej działanie polega na wywołaniu prywatnych metod klasy w celu zapisu danych dotyczących poszczególnych wskaźników. Jako przykład metody zapisującej podano 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.

# Testowanie i uruchamianie

## Model wzorcowy

Mając na uwadze to, że wymagania stawiane aplikacji są zrozumiałe i nie powinny zmienić się przez czas życia projektu oraz to, że sam projekt nie jest duży zdecydowano się na prowadzenie go kierując się modelem V[[20]](#footnote-20).



Rysunek ‑ Wizualizacja modelu V, źródło[[21]](#footnote-21)

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

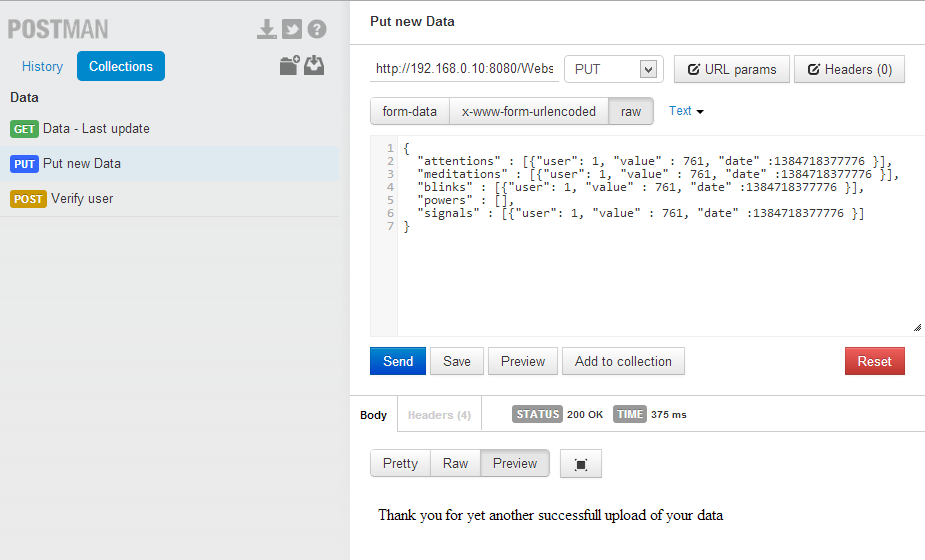
## Testowanie aplikacji mobilnej

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).

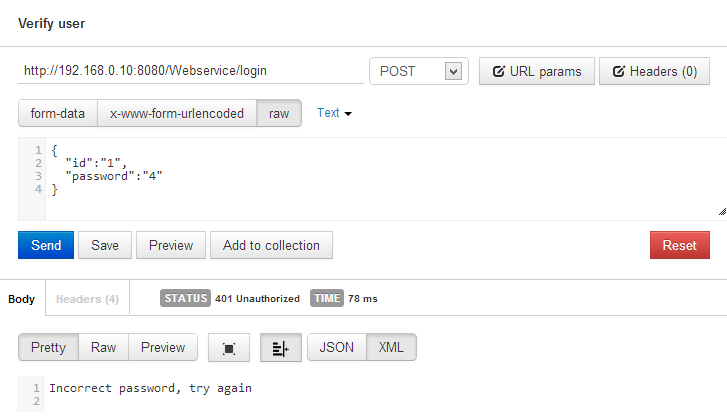
## Testowanie aplikacji serwerowej

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 Postman[[22]](#footnote-22).

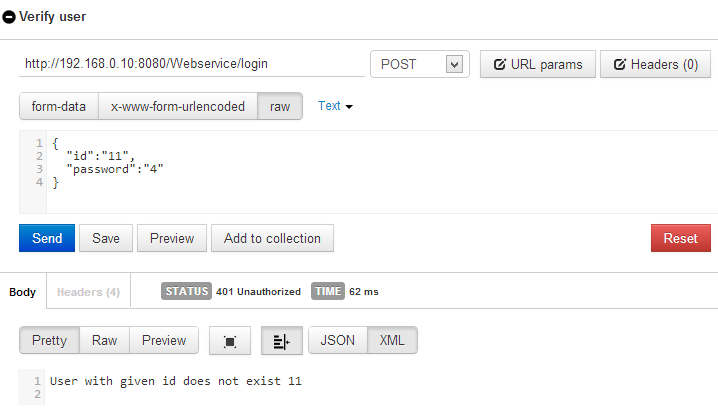
Na kolejnych stronach przedstawiono kilka zapytań z serii testów akceptacyjnych interfejsów udostępnianych aplikacji mobilnej, generujących tak pozytywne, jak  
i negatywne przypadki.



Rysunek ‑ Zrzut ekranu pokazujący program użyty do testowania części serwerowej



Rysunek ‑ Odpowiedź serwera w przypadku braku autoryzacji (podane hasło jest niepoprawne)

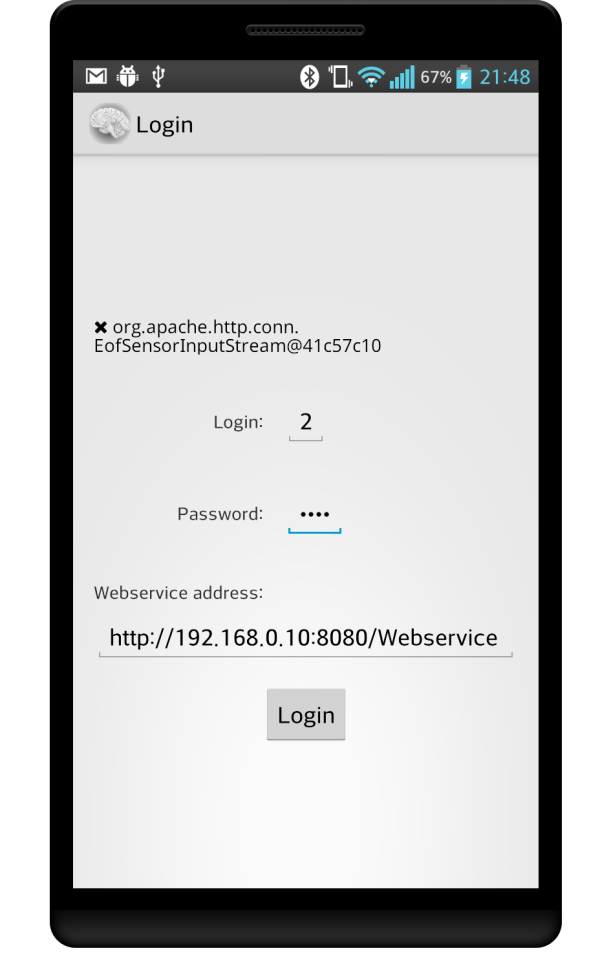


Rysunek ‑ Odpowiedź serwera w przypadku podania id dla nieistniejącego użytkownika



Rysunek ‑ Podanie niepoprawnego id użytkownika (nieistniejącego) powoduje odrzucenie zapytania

## Uruchamianie aplikacji pracujących wspólnie

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.

Rysunek ‑ Manifestacja błędu odkrytego   
w fazie uruchamiania współpracujących aplikacji

# Specyfikacja zewnętrzna

## Aplikacja mobilna

### 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.

#### Nieznane źródła

Kroki zezwolenia na instalację aplikacji z nieznanych źródeł 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 nieznanych źródeł należy wejść w ustawienia systemu, wybrać kategorię zabezpieczenia (Rysunek 6‑1) oraz zaznaczyć pole „Nieznane źródła” (Rysunek 6‑2).

#### Z linii poleceń

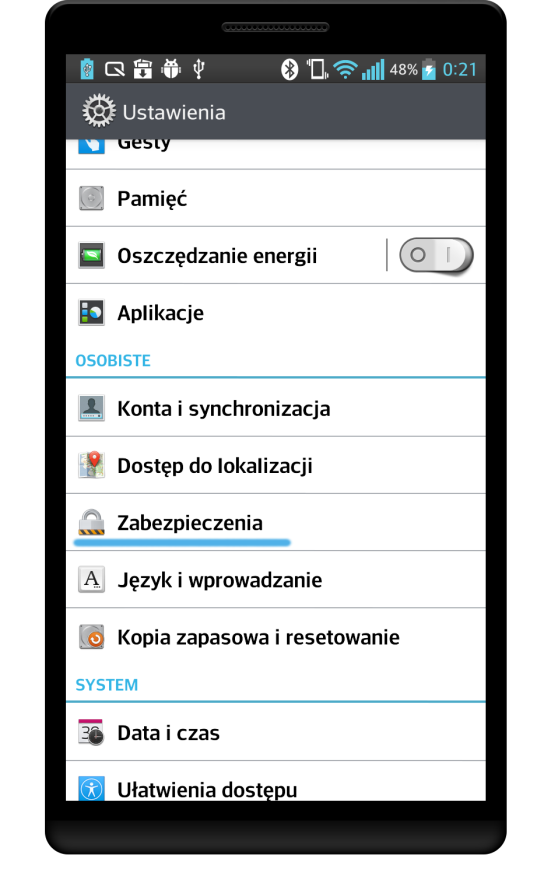
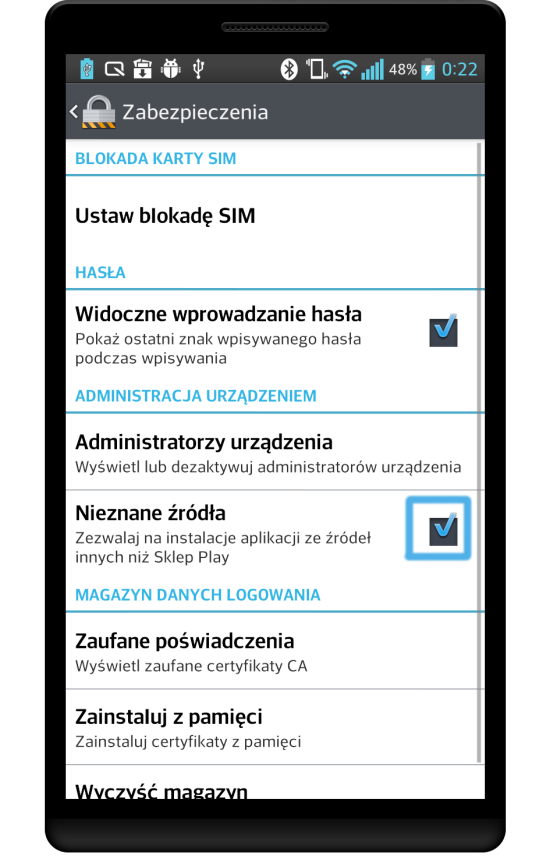
Podane instrukcje dotyczą systemu Windows i zakładają podpięcie jednego urządzenia fizycznego do systemu.

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.

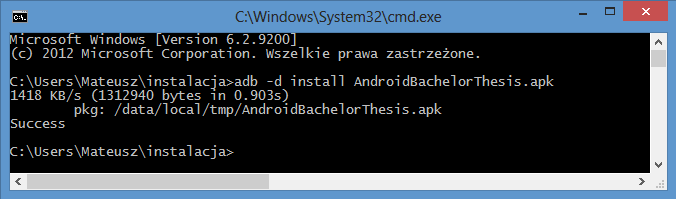
Należy najpierw wykonać kroki opisane w rozdziale 6.1.1.1, następnie otworzyć konsolę w folderze w którym znajduje się plik .apk z aplikacją i wydać polecenie:

adb -d install AndroidBachelorThesis.apk

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

|  |  |
| --- | --- |
| Rysunek ‑ Widok ustawień systemu | Rysunek ‑ Opcja, którą należy  zaznaczyć aby móc zainstalować aplikację |

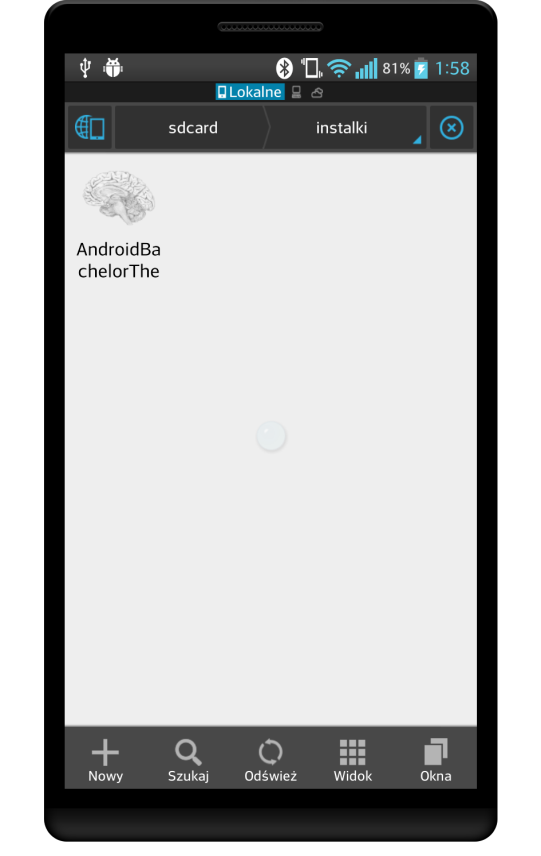
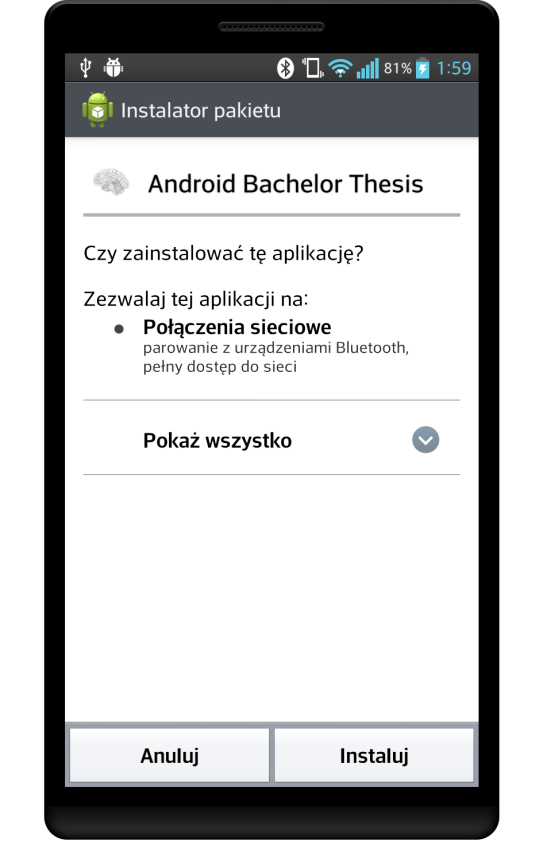


Rysunek ‑ Polecenie instalacji aplikacji na urządzeniu

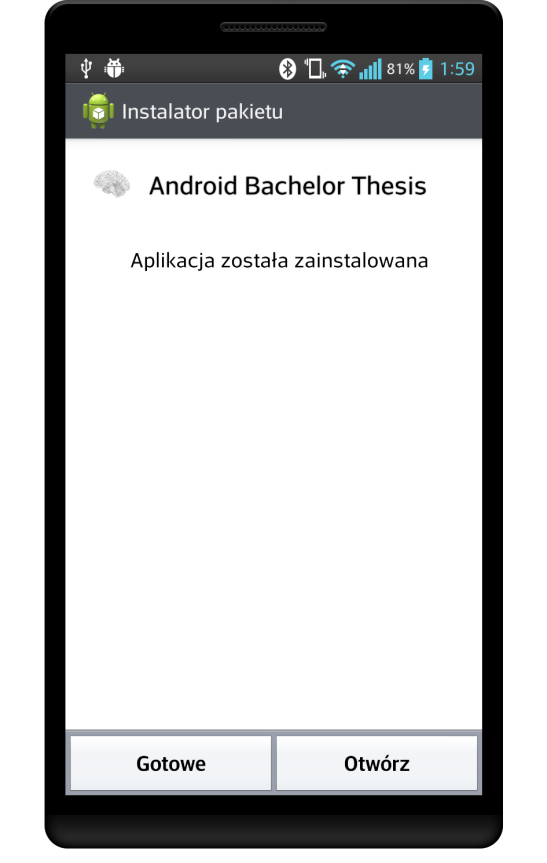
#### Poprzez eksplorator plików

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 Explorer[[23]](#footnote-23)) 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 instalację (Rysunek 6‑3). Po poprawnej instalacji zostanie wyświetlony ekran potwierdzający prawidłowe zainstalowanie programu (Rysunek 6‑6).

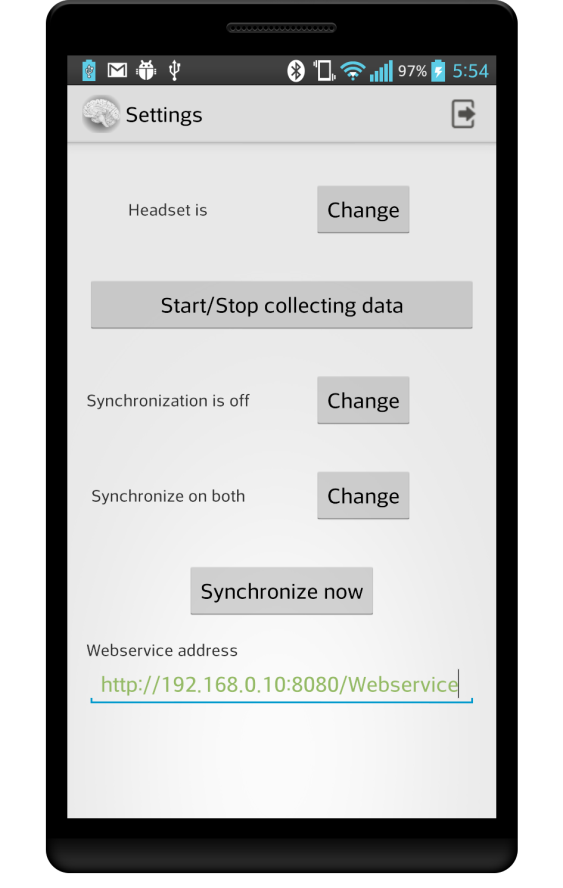
|  |  |
| --- | --- |
| Rysunek ‑ Widok eksploratora plików ES File Explorer | Rysunek ‑ Monit z potwierdzeniem instalacji |



Rysunek ‑ Potwierdzenie pomyślnie przeprowadzonej instalacji

### 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.

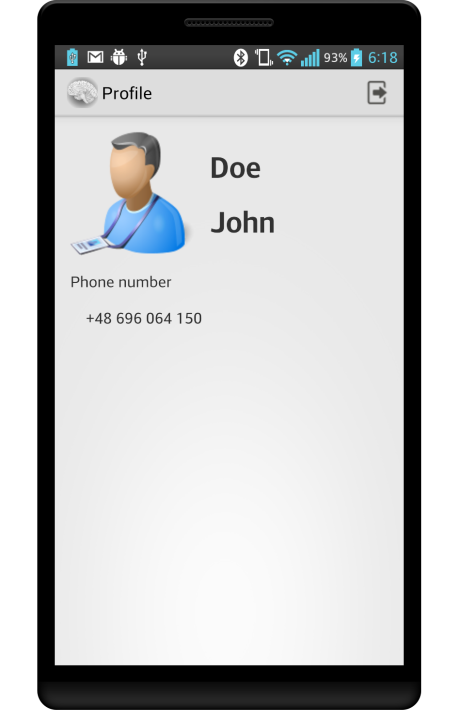
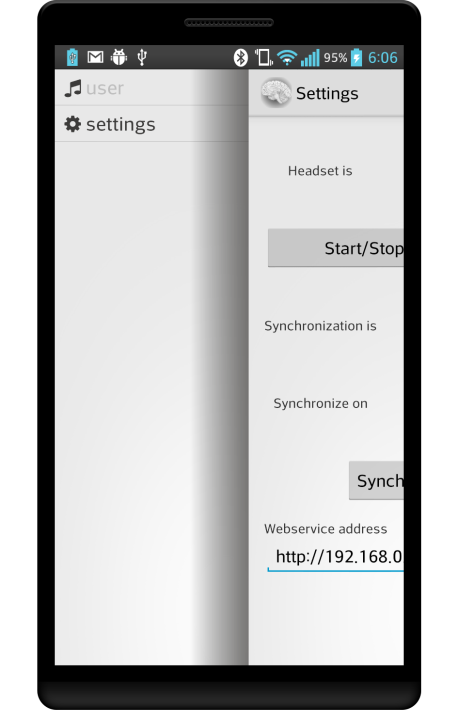
Po poprawnym zalogowaniu użytkownikowi wyświetlony jest główny ekran aplikacji (rysunek 6-8). Ekran ten udostępnia możliwość sterowania momentem synchronizacji danych z serwerem i ustawiania adresu serwera. Do funkcji dotyczących zarządzania zestawem EEG dostępnych z tego poziomu należy zaliczyć: ustanawianie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

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 ‑ Główny ekran aplikacji



## Aplikacja serwerowa

Rysunek ‑ Ekran profilu

Rysunek ‑ Menu aplikacji

### Instalacja

#### MySQL

Aplikacja serwerowa do poprawnego działania wymaga zewnętrznej bazy MySQL, jej instalator można pobrać z podstrony projektu[[24]](#footnote-24). Autor zakłada, że folder zawierający[[25]](#footnote-25) 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.

#### Glassfish

Po pobraniu serwera Glassfish należy go rozpakować i dodać folder bin z katalogu do zmiennej PATH. Następnie skopiować załączoną do pracy bibliotekę MySQL Connector[[26]](#footnote-26) do katalogu <scieżka\_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.

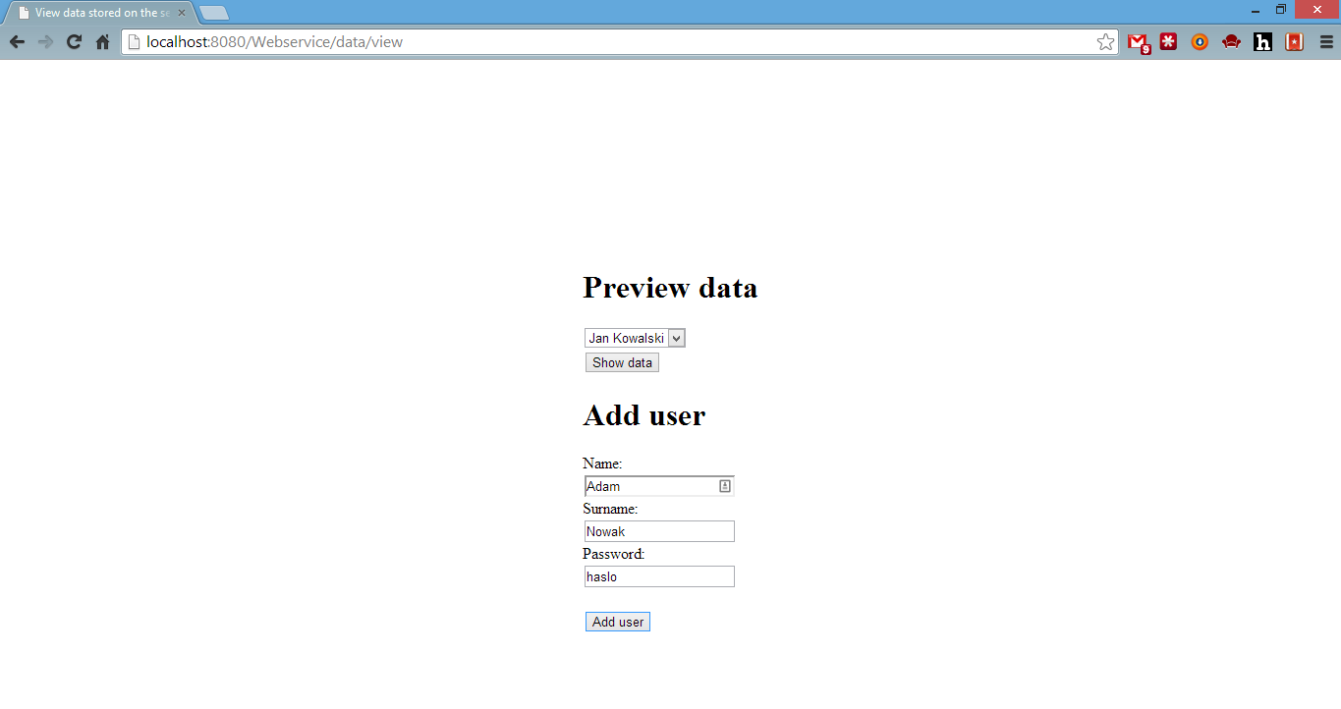
### Użycie

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

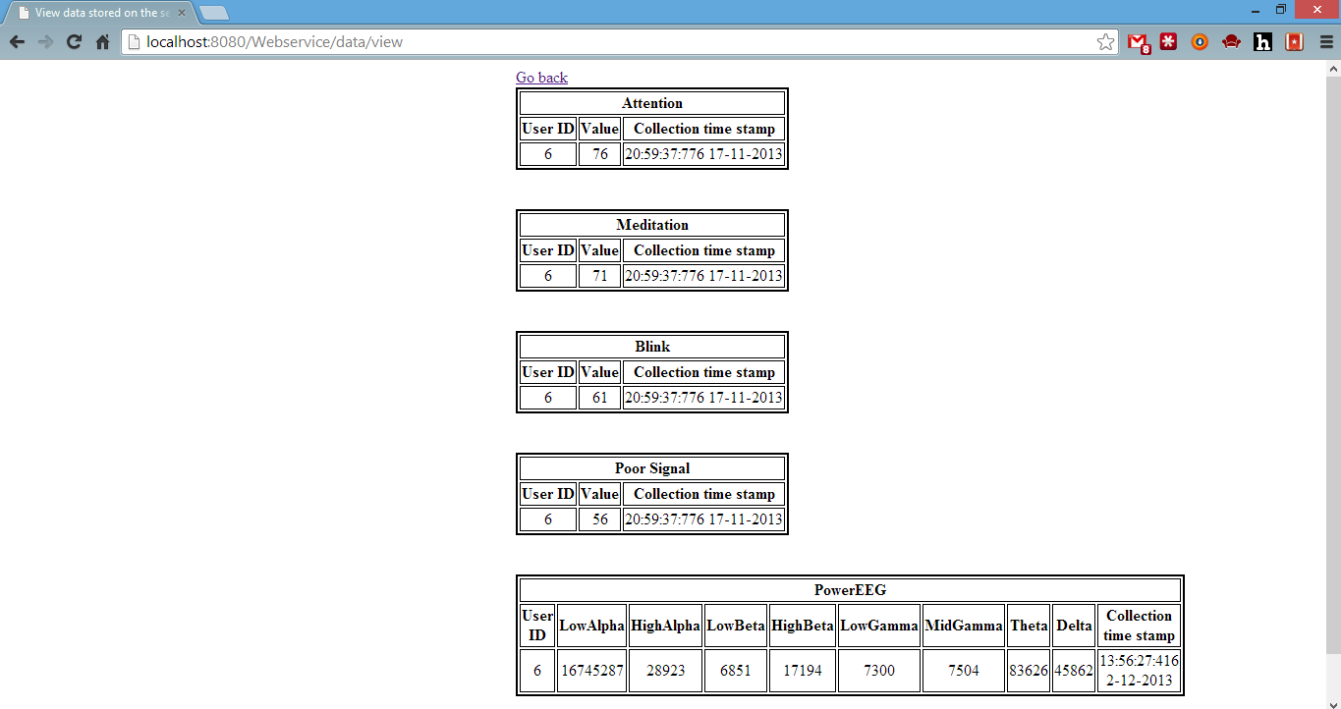
Aby wykonać obie wymienione czynności należy rozpocząć od odwiedzenia strony /view.jsp. Na tej stronie jest wystawiany punkt logowania administratora  
(rysunek 6-11). Domyślne dane do logowania to użytkownik admin z hasłem concept. Po zalogowaniu dostępne są dwie akcje: wyświetlanie danych dla danego 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.

Rysunek ‑ Punkt logowania administratora

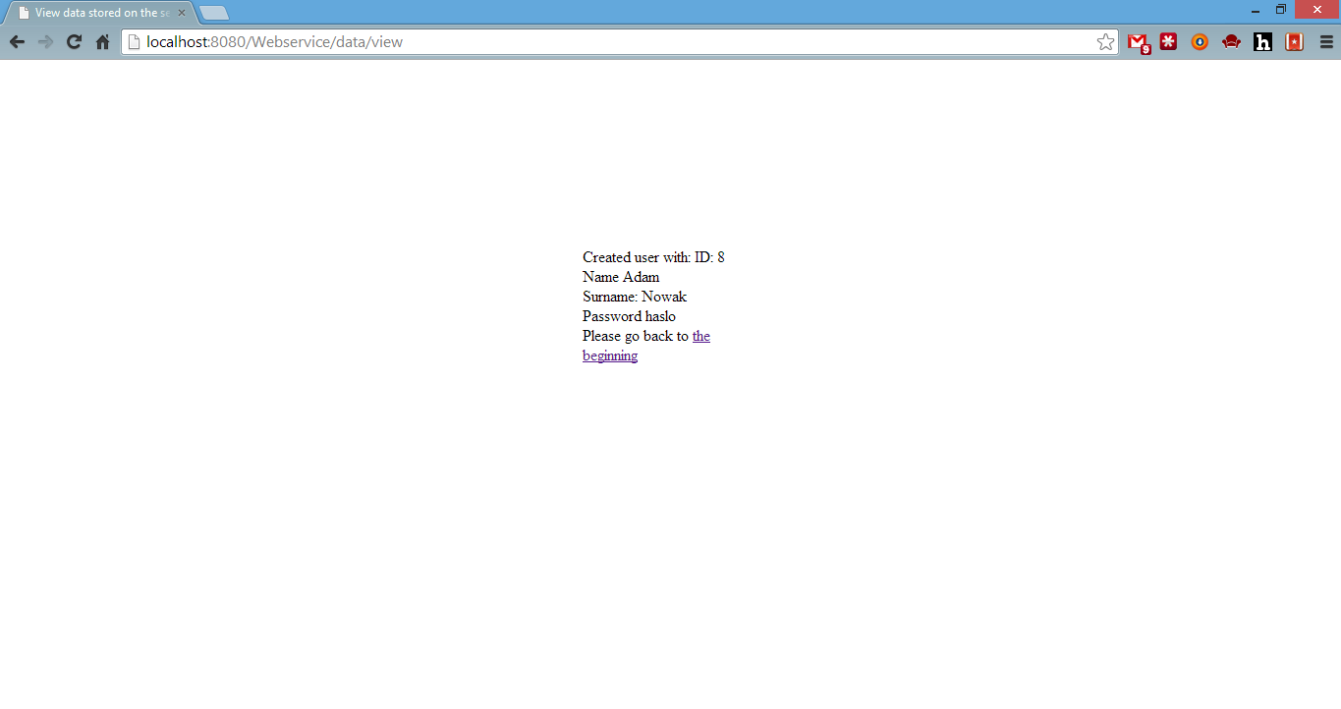
Aby dodać użytkownika na stronie widocznej zaraz po zalogowaniu, należy wypełnić wszystkie pola, a następnie kliknąć przycisk „Add user”. Spowoduje to przeładowanie strony i, w zależności od poprawności podanych danych (pola nie mogą być puste), pokazanie odpowiedniego komunikatu. Przykładowy komunikat wyświetlony po poprawnym dodaniu użytkownika jest widoczny na rysunku 6-14, a po podaniu niepoprawnych danych na rysunku 6-15.



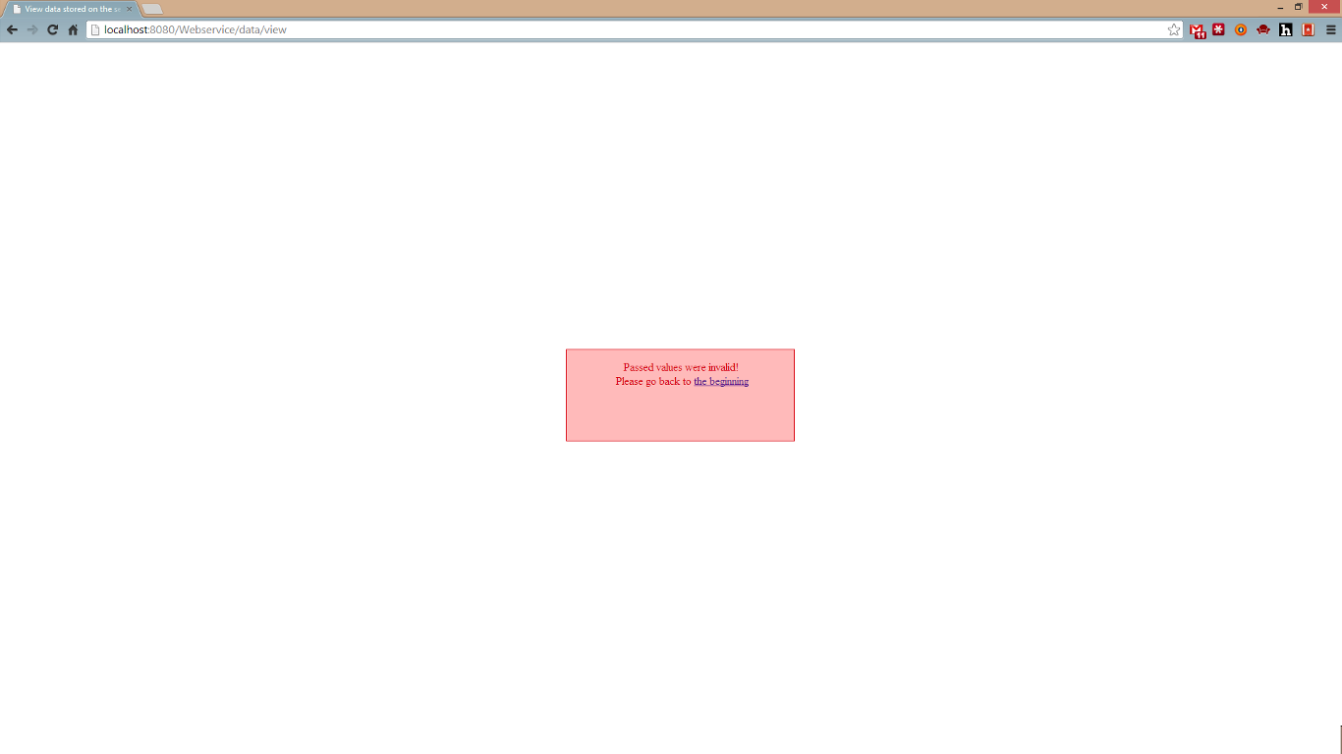
Rysunek ‑ Dostępne akcje



Rysunek ‑ Lista danych użytkownika, tutaj zawierająca tylko po jednej wartości dla każdego ze wskaźników



Rysunek ‑ Komunikat potwierdzający poprawne dodanie użytkownika



Rysunek ‑ Komunikat informujący o niepoprawnie wypełnionym formularzy rejestracyjnym

# 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 użytkowania).

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 użyć 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.

# 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 praktycetłumaczył 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]

# Spis rysunków

[Rysunek 2‑1 Koncepcja wzorca model widok kontroler, źródło 2](#_Toc377488136)

[Rysunek 2‑2 Zestawienie głównych klas fal mózgowych, źródło [2], strona 11 4](#_Toc377488137)

[Rysunek 2‑3 Rozmieszczenie elektrod w systemie 10-20, źródło 4](#_Toc377488138)

[Rysunek 2‑4 Architektura systemu, źródło własne. 5](#_Toc377488139)

[Rysunek 3‑1 Podział aplikacji mobilnej na procesy 7](#_Toc377488140)

[Rysunek 3‑2 Struktura klas w procesie akwizycji danych 8](#_Toc377488141)

[Rysunek 3‑3 Struktura klas uczestniczących w autoryzacji użytkownika w procesie danych 9](#_Toc377488142)

[Rysunek 3‑4 Diagram sekwencji pokazujący realizację zadania autoryzacji użytkownika wewnątrz procesu danych 10](#_Toc377488143)

[Rysunek 3‑5 Struktura klas części procesu odpowiedzialnego za obsługę danych 10](#_Toc377488144)

[Rysunek 3‑6 Klasy biorące udział w synchronizacji danych 11](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488145)

[Rysunek 3‑7 Klasy wykonujące komunikację z procesem akwizycji 12](#_Toc377488146)

[Rysunek 3‑8 Łącznik udostępniający funkcję autoryzacji 13](#_Toc377488147)

[Rysunek 3‑9 Klasy łącznika danych 14](#_Toc377488148)

[Rysunek 3‑10 Klasy łącznika komend 15](#_Toc377488149)

[Rysunek 3‑11 Schemat bazy danych 19](#_Toc377488150)

[Rysunek 4‑1 Zasada działania biblioteki greenDAO, źródło strona projektu 21](#_Toc377488151)

[Rysunek 5‑1 Wizualizacja modelu V, źródło 28](#_Toc377488152)

[Rysunek 5‑2 Zrzut ekranu pokazujący program użyty do testowania części serwerowej 29](#_Toc377488153)

[Rysunek 5‑3 Odpowiedź serwera w przypadku braku autoryzacji (podane hasło jest niepoprawne) 30](#_Toc377488154)

[Rysunek 5‑4 Odpowiedź serwera w przypadku podania id dla nieistniejącego użytkownika 30](#_Toc377488155)

[Rysunek 5‑5 Podanie niepoprawnego id użytkownika (nieistniejącego) powoduje odrzucenie zapytania 31](#_Toc377488156)

[Rysunek 5‑6 Manifestacja błędu odkrytego w fazie uruchamiania współpracujących aplikacji 31](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488157)

[Rysunek 6‑1 Widok ustawień systemu 33](#_Toc377488158)

[Rysunek 6‑2 Opcja, którą należy zaznaczyć aby móc zainstalować aplikację 33](#_Toc377488159)

[Rysunek 6‑3 Polecenie instalacji aplikacji na urządzeniu 33](#_Toc377488160)

[Rysunek 6‑4 Widok eksploratora plików ES File Explorer 34](#_Toc377488161)

[Rysunek 6‑5 Monit z potwierdzeniem instalacji 34](#_Toc377488162)

[Rysunek 6‑6 Potwierdzenie pomyślnie przeprowadzonej instalacji 34](#_Toc377488163)

[Rysunek 6‑7 Ekran logowania 35](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488164)

[Rysunek 6‑8 Główny ekran aplikacji 35](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488165)

[Rysunek 6‑9 Ekran profilu 36](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488166)

[Rysunek 6‑10 Menu aplikacji 36](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488167)

[Rysunek 6‑11 Punkt logowania administratora 37](file:///C:\inzynierka\documentation-bachelor-thesis\Projekt%20Inzynierski.docx#_Toc377488168)

[Rysunek 6‑12 Dostępne akcje 38](#_Toc377488169)

[Rysunek 6‑13 Lista danych użytkownika, tutaj zawierająca tylko po jednej wartości dla każdego ze wskaźników 38](#_Toc377488170)

[Rysunek 6‑14 Komunikat potwierdzający poprawne dodanie użytkownika 39](#_Toc377488171)

[Rysunek 6‑15 Komunikat informujący o niepoprawnie wypełnionym formularzy rejestracyjnym 39](#_Toc377488172)

# Spis fragmentów kodu

[Listing 3‑1 Format zapytania weryfikującego tożsamość użytkownika 16](#_Toc377488173)

[Listing 3‑2 Format odpowiedzi zawierającej ostatnie czasy aktualizacji 17](#_Toc377488174)

[Listing 3‑3 Format danych przesyłanych do serwera w celu utrwalenia 17](#_Toc377488175)

[Listing 3‑4 Obiekt JSON reprezentujący wartości prostego wskaźnika 17](#_Toc377488176)

[Listing 3‑5 Obiekt JSON reprezentujący wartości złożonego wskaźnika 18](#_Toc377488177)

[Listing 4‑1 Nagłówek i pola klasy MainService 22](#_Toc377488178)

[Listing 4‑2 Wybrane fragmenty klasy EEGAcquisitionService 24](#_Toc377488179)

[Listing 4‑3 Metoda udostępniająca połączenie z bazą danych 26](#_Toc377488180)

[Listing 4‑4 Fragment klasy dokonującej odczytu z bazy danych 26](#_Toc377488181)

[Listing 4‑5 Fragment klasy zapisującej dane do bazy danych 27](#_Toc377488182)

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

1. Definicja podana za <http://www.json.org/json-pl.html>, [dostęp 25 grudnia 2013]. [↑](#footnote-ref-1)
2. Strona domowa Trygve H. M. Reenskaug <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> [dostęp 25 grudnia 2013] [↑](#footnote-ref-2)
3. Wiki wydziału medycznego Uniwersytetu Michigan <https://wiki.umms.med.umich.edu/download/attachments/90734989/EEG+10-20+system+map.JPG> [↑](#footnote-ref-3)
4. <http://archive.eclipse.org/eclipse/downloads/drops4/R-4.3.1-201309111000/> [↑](#footnote-ref-4)
5. <https://glassfish.java.net/download.html> [↑](#footnote-ref-5)
6. <http://developer.android.com/sdk/installing/studio.html> [↑](#footnote-ref-6)
7. Strona na której dostępne jest więcej informacji na temat pluginu i jego użycia

   <https://github.com/jvoegele/gradle-android-plugin/wiki> [↑](#footnote-ref-7)
8. Więcej na temat serwisów można przeczytać na stronie <http://developer.android.com/guide/components/services.html> [↑](#footnote-ref-8)
9. Na temat AIDL więcej informacji można zasięgnąć na stronie

   <http://developer.android.com/guide/components/aidl.html> [↑](#footnote-ref-9)
10. Strona projektu znajduje się pod adresem <http://jakewharton.github.io/butterknife/>, zawiera opis i kompletne wprowadzenie do użycia. [↑](#footnote-ref-10)
11. Strona domowa projektu może zostać znaleziona pod adresem <http://square.github.io/dagger/> i również zawiera wprowadzenie do użycia biblioteki. [↑](#footnote-ref-11)
12. Specyfikacja dostępna pod adresem <https://jcp.org/aboutJava/communityprocess/final/jsr330/> [↑](#footnote-ref-12)
13. Strona projektu <https://code.google.com/p/roboguice/> [↑](#footnote-ref-13)
14. Strona domowa projektu <https://github.com/jfeinstein10/SlidingMenu> [↑](#footnote-ref-14)
15. Strona domowa projektu zawierająca liczne wprowadzenia i dokumentację znajduje się pod adresem <http://greendao-orm.com/documentation/introduction/> [↑](#footnote-ref-15)
16. Do pobrania za darmo ze strony <http://store.neurosky.com/products/developer-tools-3-android> [↑](#footnote-ref-16)
17. Dostępny do pobrania ze strony projektu: <http://dev.mysql.com/downloads/connector/j/> [↑](#footnote-ref-17)
18. Strona projektu dostępna pod adresem <http://commons.apache.org/> [↑](#footnote-ref-18)
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> [↑](#footnote-ref-19)
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] [↑](#footnote-ref-20)
21. <http://www.the-software-experts.de/e_dta-sw-process.htm> [↑](#footnote-ref-21)
22. Strona projektu dostępna w sieci pod adresem <http://www.getpostman.com/> [↑](#footnote-ref-22)
23. Adres pod którym program jest dostępny do pobrania:

    <https://play.google.com/store/apps/details?id=com.estrongs.android.pop&hl=pl> [↑](#footnote-ref-23)
24. adres <http://dev.mysql.com/downloads/installer/> [↑](#footnote-ref-24)
25. Jest to folder MySQL Workbench CE z katalogu instalacyjnego. [↑](#footnote-ref-25)
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/> [↑](#footnote-ref-26)