Politechnika Poznańska
Wydział Elektryczny
Kierunek Informatyka

KakaduVoIP

Projekt zespołowy
Telefonia IP

Autorzy:

Szymon Zieliński
szymon.r.zielinski@gmail.com
126812

Oskar Rutkowski
oskar.rutkowski@student.put.poznan.pl
126845

6 maja 2018

Spis treści

1 Charakterystyka ogólna projektu

Aplikacja pozwala na komunikowanie się użytkowników za pomocą mikrofonu oraz głośników w czasie rzeczywistym. Aplikacja jest stworzona w języku Java oraz działa w oparciu o architekturę klient-serwer. Użytkownik po uruchomieniu, jest proszony o podanie nazwy użytkownika oraz adresu serwera do którego chce się połączyć. Będąc połączonym, użytkownik ma możliwość stworzenia własnego pokoju konferencyjnego lub dołączenia do istniejącego już pokoju. Aby połączyć się z pokojem wybranym z listy, użytkownik musi podać hasło pokoju. Po podaniu prawidłowego hasła, użytkownik ma możliwość rozmowy przy pomocy naciśniętego przycisku lub zmianę metody komunikacji na opcję mówienia ciągłego. Użytkownik ma możliwość wyciszenia swojego mikrofonu lub głośników, jak też innego użytkownika, znajdującego się w pokoju.

2 Architektura systemu

System KakaduVoIP jest oparty o model architektury klient-serwer. Model ten umożliwia podział zadań, serwer zapewnia usługi dla klientów, a klienci zgłaszają żądania obsługi. Serwer składuje informacje o pokojach w plikach lokalnych.

3 Wymagania

W tym paragrafie zostaną omówione wymagania aplikacji funkcjonalne oraz niefunkcjonalne.

3.1 Wymagania funkcjonalne

- użytkownik łączy się z systemem jedynie za pomocą nazwy użytkownika oraz adresu serwera,
- użytkownik może zmieniać ustawienia aplikacji względem możliwości komunikowania się, tj. mówienie przez naciśnięcie przycisku lub mówienie ciągłe,
- użytkownik może utworzyć pokój konferencyjny, nadając mu nazwę i hasło dostępu do pokoju oraz tworząc hasło administracyjne, stając się administratorem tego pokoju,
- administrator pokoju, który stworzył pokój, może usunąć pokój konferencyjny,
- administrator, który założył pokój, może wyrzucić innych użytkowników z pokoju.
- użytkownik może dołączyć do pokoju wybranego z listy aktualnie dostępnych pokoi na serwerze, podając hasło dostępu do pokoju,
- użytkownik w pokoju konferencyjnym może opuścić pokój,
- użytkownik w pokoju konferencyjnym może wyciszyć innego użytkownika, aby go nie słyszeć,
- użytkownik w pokoju konferencyjnym może komunikować się z innymi użytkownikami za pomocą przycisku lub mówienia ciągłego,
- użytkownik w pokoju konferencyjnym może wyciszyć swój mikrofon lub wyłączyć używanie głośników,

3.2 Wymagania niefunkcjonalne

- nowo utworzony pokój nie potrzebuje administratora pokoju będącego aktualnie w pokoju,
- pokój może być usunięty przez administratora pokoju dzięki podaniu hasła dostępu do pokoju oraz hasła administracyjnego,
- po usunięciu pokoju przez administratora pokoju, wszyscy użytkownicy zostają automatycznie przekierowani do głównej strony aplikacji,
- jeden użytkownik może być administratorem tylko jednego pokoju,

- długo nieużywane pokoje zostają automatycznie usunięte,
- aplikacja klienta oraz serwer napisane w języku Java,
- aplikacja wieloplatformowa,
- szyfrowanie dźwięku,
- szyfrowanie zapytań do serwera,
- do połączenia z serwerem nie jest potrzebna rejestracja użytkownika, jedynym wymogiem jest podanie nazwy użytkownika oraz adres serwera,
- do składowania haseł zabezpieczających pokoje, posłużą pliki lokalne składowane na serwerze,
- hasła dostępu do pokoju oraz hasła administracyjne będą poddane funkcji haszującej,
- interfejs stworzony w języku polskim,
- do sprawnego działania aplikacji, niezbędne są słuchawki lub głośniki oraz mikrofon.

4 Narzędzia, środowiska, biblioteki, kodeki

W niniejszym paragrafie zostaną opisane wykorzystane w aplikacji narzędzia, środowiska, biblioteki oraz kodeki.

Głównym językiem wykorzystanym do napisania aplikacji jest język Java w wersji 8 Update 162. Wykorzystana jest technika Maven Project, dzięki której możliwa jest praca w różnych środowiskach programistycznych takich jak Intellij oraz Eclipse Oxygen.

Wykorzystane frameworki oraz biblioteki:

- Apache MINA - Open Source Framework, służący do budowania aplikacji sieciowych,
- Java Media Framework - technologia umożliwiająca wstawianie multimediów do aplikacji napisanych w języku Java oraz transmisję audio za pomocą protokołu RTP,
- Protocol Buffers - narzędzie do tworzenia własnych protokołów,
- java.security.MessageDigest - biblioteka wykorzystana do nakładania funkcji skrótu na hasła,
- javax.security oraz javax.crypto - biblioteki wykorzystane do szyfrowania.

Hasła wraz z potrzebnymi informacjami o pokojach oraz użytkownikach są przetrzymywane w plikach lokalnych przechowywanych po stronie serwera.

Użyte kodeki, zawarte w JMF:
MPEG Layer II Audio (.mp2) [MPEG layer 2 audio]

5 Protokoły

Jednymi z najważniejszych protokołów użytych są:

- Warstwa aplikacji:
  - RTP - protokół transmisji w czasie rzeczywistym - do przesyłania dźwięku,
  - RTCP (ang. Real Time Control Protocol) - protokół sterujący używany do okresowej transmisji pakietów kontrolnych do wszystkich uczestników sesji. Pozwala monitorować dostarczanie danych przez protokół RTP i transport zwrotnej informacji odnośnie jakości transmisji,

- protokoły własne - protokoły do komunikacji z serwerem:
- protokół informacji o pokoju:

  message Room {
  required string room_name = 1;
  required string owner = 2;
  }

  Protokół informacji o pokoju zawiera niezbędne pola do identyfikacji kontkretnych pokoi, takie jak nazwa pokoju oraz właściciel pokoju (użytkownik, który stworzył konkretny pokój).
- protokół żądania klienta podłączenia do serwera:

  message LoginToServerRequest {
  required string nick = 1;
  }

  Protokół żądania klienta podłączenia do serwera zawiera pole nick, wskazujące na nazwę użytkownika, chcącego podłączyć się do serwera.
- protokół odpowiedzi na żądanie klienta podłączenia do serwera:

  message LoginToServerResponse {
  required StatusCode status = 1;
  repeated Room roomList = 2;
  }

  Protokół odpowiedzi, zwraca klientowi status żądania oraz listę dostępnych pokoi, przy poprawnym połączeniu się z serwerem.
- protokół opuszczenia serwera:
  message LeaveServerRequest{
  required string nick = 1;
  }

  Protokół opuszczenia serwera zawiera pole nick, wskazujący na użytkownika, który chce opuścić serwer.
- protokół odpowiedzi na opuszczenie serwera:
  message LeaveServerResponse {
  required StatusCode status = 1;
  }

  Protokół odpowiedzi zawiera pole status, informujące o pomyślnej lub negatywnej odpowiedzi na żądanie.
- protokół żądania zadrządzania pokojem:
  message ManageRoomRequest {
  required ManageRoomEnum manageRoomEnum = 1;
  required string nick = 2;
  required string roomName = 3;
  optional string password = 4;
  optional string adminPassword = 5;
  optional string otherUserNick = 6;
  }

  Protokół zarządzania pokojem służy do zarządzania pokojem. Niezbędne pola manageRoomEnum, nick oraz roomName wskazują kolejno na: akcje związane z zarządzaniem pokojem, nazwę użytkownika, który chce zarządzać oraz nazwę pokoju, który ma być zarządzany.

Dodatkowe pola wskazują na hasło dołączenia do pokoju, hasło administracyjne, służące do usuwania pokoju przez administratora oraz nazwa innego użytkownika, w przypadku wyciszania lub usuwania z pokoju.

- protokół odpowiedzi na żądanie zarządzania pokojem: message ManageRoomResponse {
  required StatusCode status = 1;
  optional string keyToRoom = 2;
  }

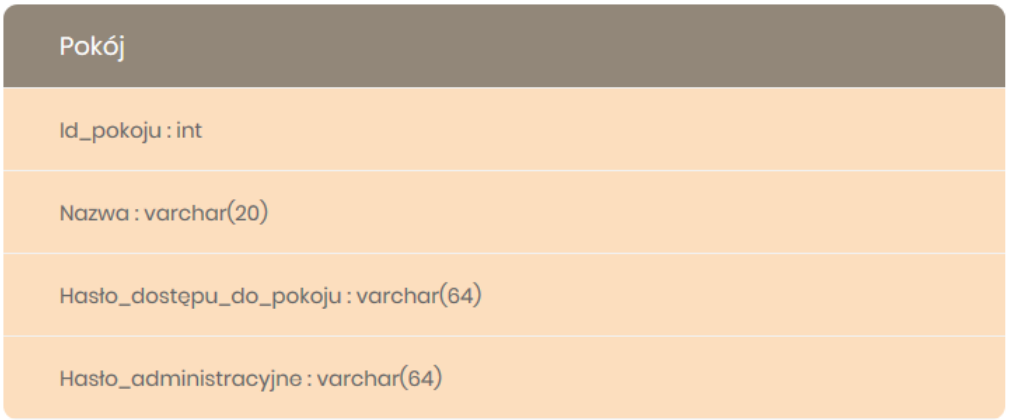
  Protokół odpowiedzi zawiera status odpowiedzi oraz opcjonalny klucz szyfrowania.
- typ wyliczeniowy do zarządzania pokojem:
  enum ManageRoomEnum {
  CREATE_ROOM = 1;
  DELETE_ROOM = 2;
  JOIN_ROOM = 3;
  LEAVE_ROOM = 4;
  MUTE_USER = 5;
  UNMUTE_USER = 6;
  KICK_USER = 7;
  }

  Typ wyliczeniowy pomocny przy protokole zarządzania pokojem.

- Warstwa transportowa:
  - TCP - protokół kontroli transmisji - do komunikacji z serwerem,
  - UDP - transportowy protokół bezpołączeniowy - do przesyłania próbek z dźwiękiem.
- Warstwa sieci:
  - IP - podstawowy protokół stosowany w Internecie.

6 Schemat bazy danych

W niniejszym paragrafie zostanie przedstawiony schemat bazy przechowującej dane pokojów.



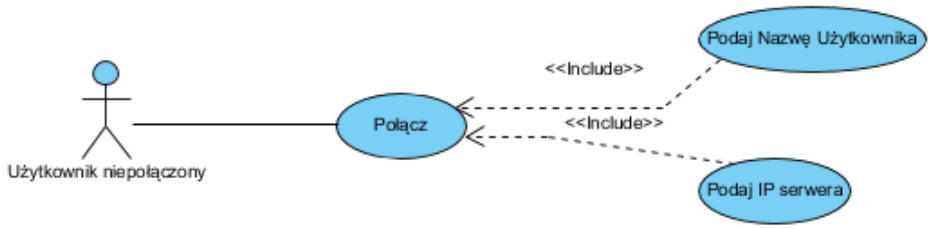
Rysunek 1: Model tabeli Pokój

Zgodnie z rysunkiem 1, baza danych składa się z jednej tabeli opisującej pokój. Id_pokoju wskazuje na identyfikator konkretnego utworzonego przez użytkownika pokoju. Nazwę pokoju nadaje użytkownik podczas tworzenia pokoju. Hasło dostępu do pokoju oraz hasło administracyjne są wynikiem funkcji skrótu.

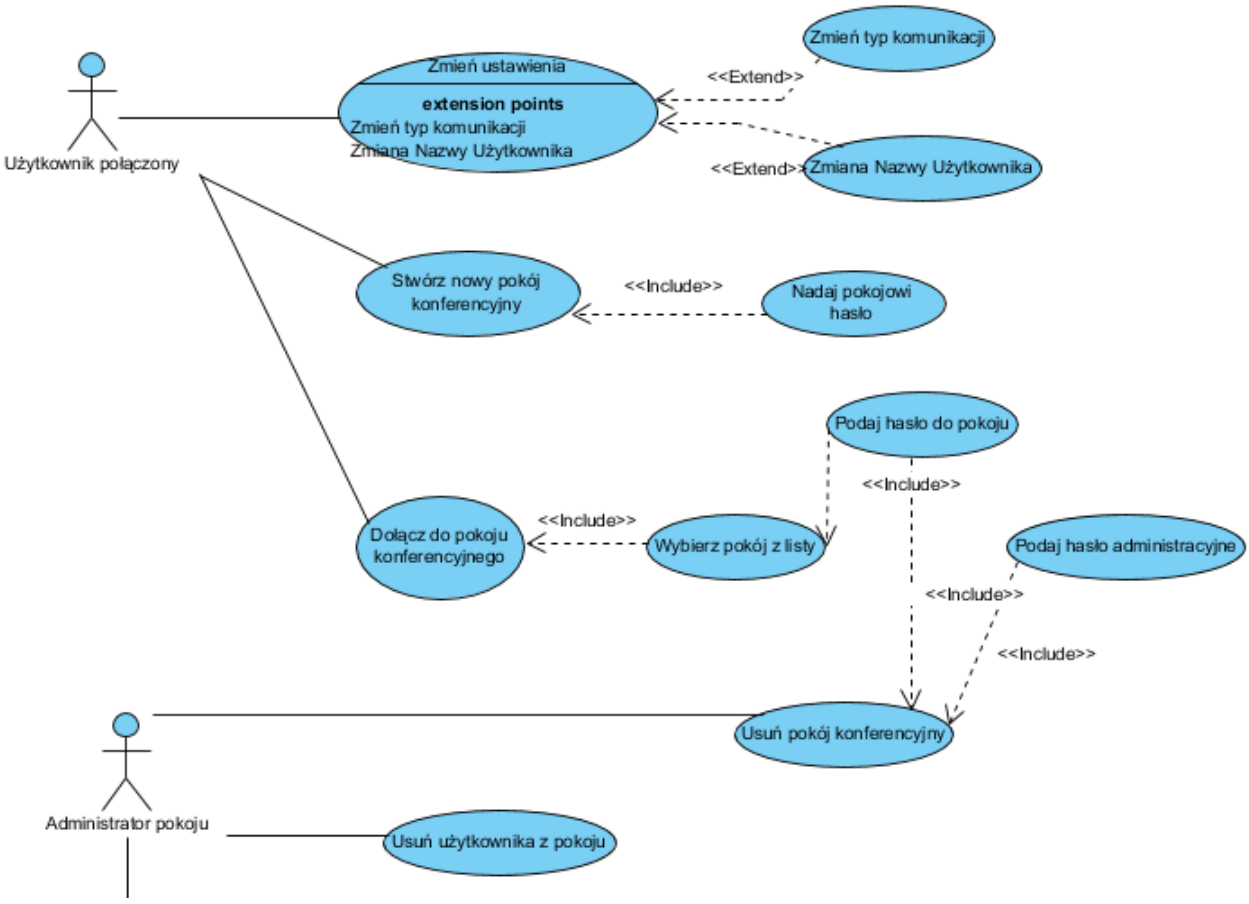
7 Diagramy UML

7.1 Diagramy przypadków użycia

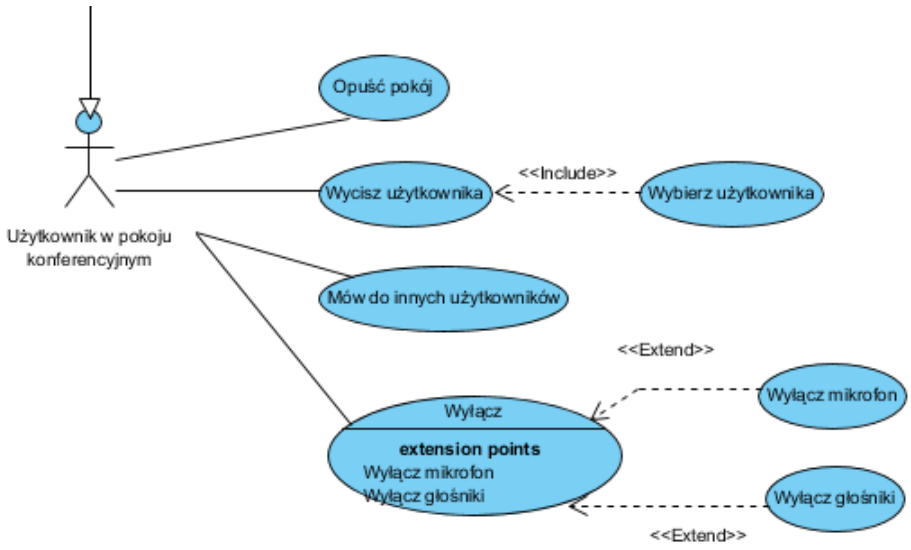
W niniejszym paragrafie zostanią przedstawione diagramy UML aktorów oraz ich zadań funkcjonalnych.



Rysunek 2: Diagram użytkownika niepołączonego



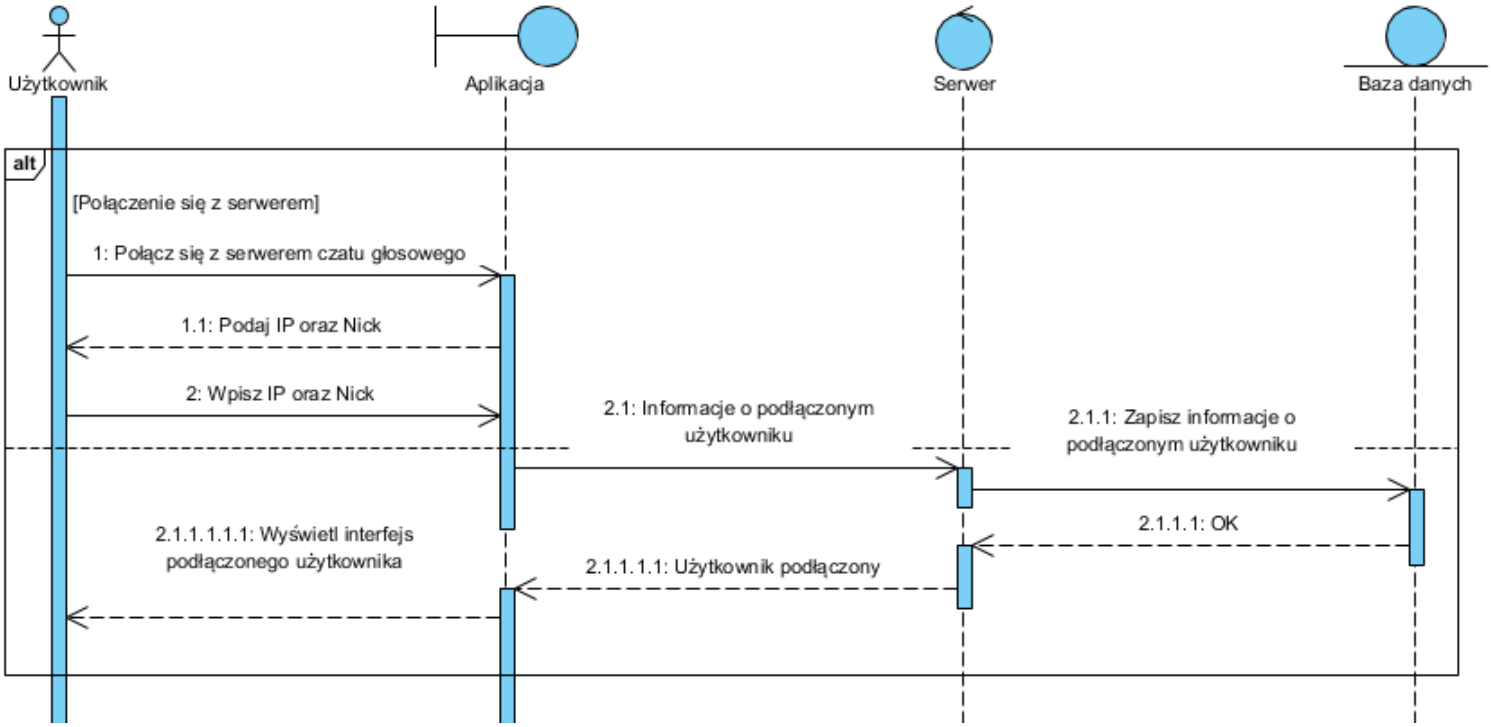
Rysunek 3: Diagram użytkownika połączonego oraz administratora pokoju



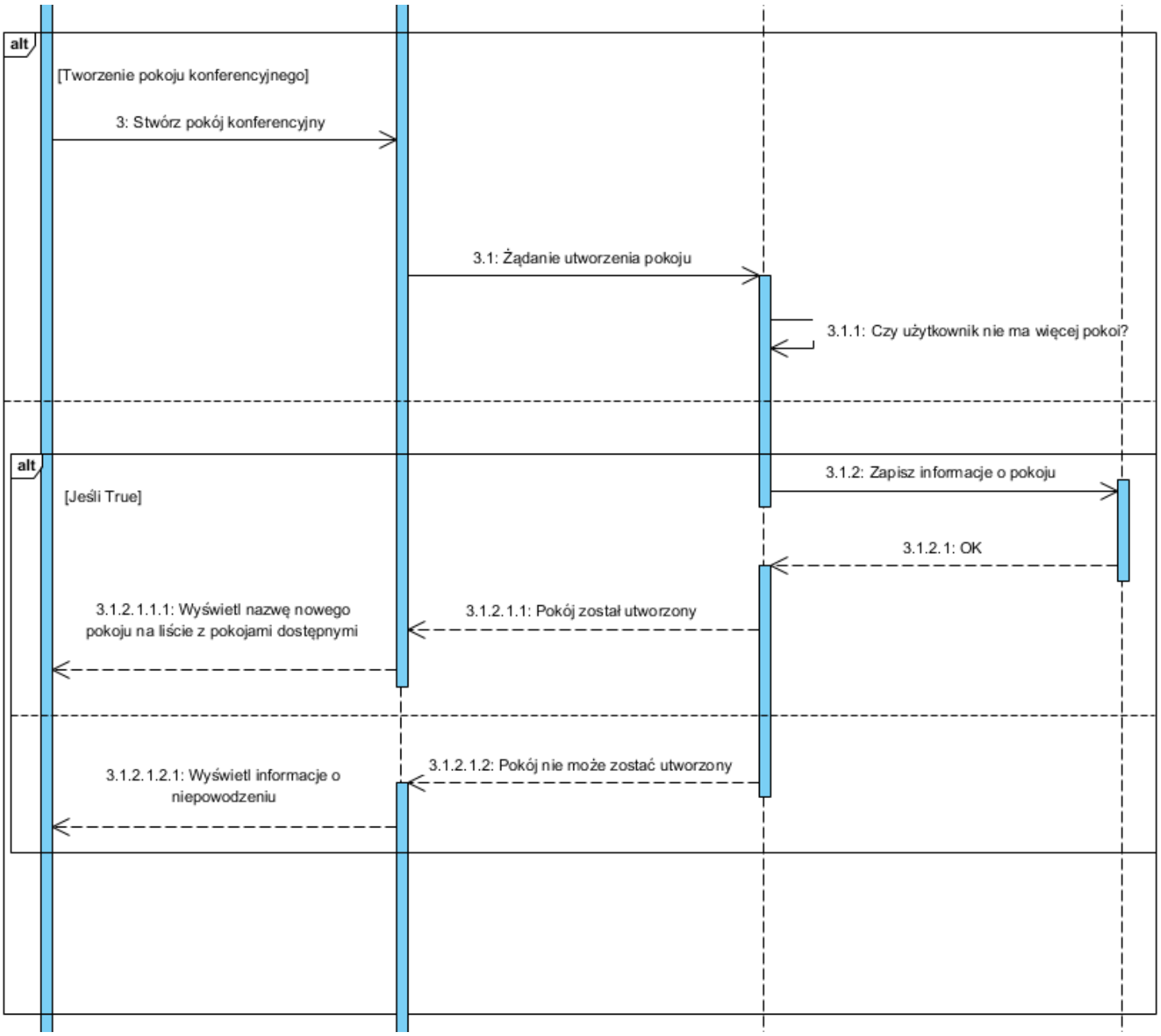
Rysunek 4: Diagram użytkownika w pokoju konferencyjnym

7.2 Diagramy sekwencji

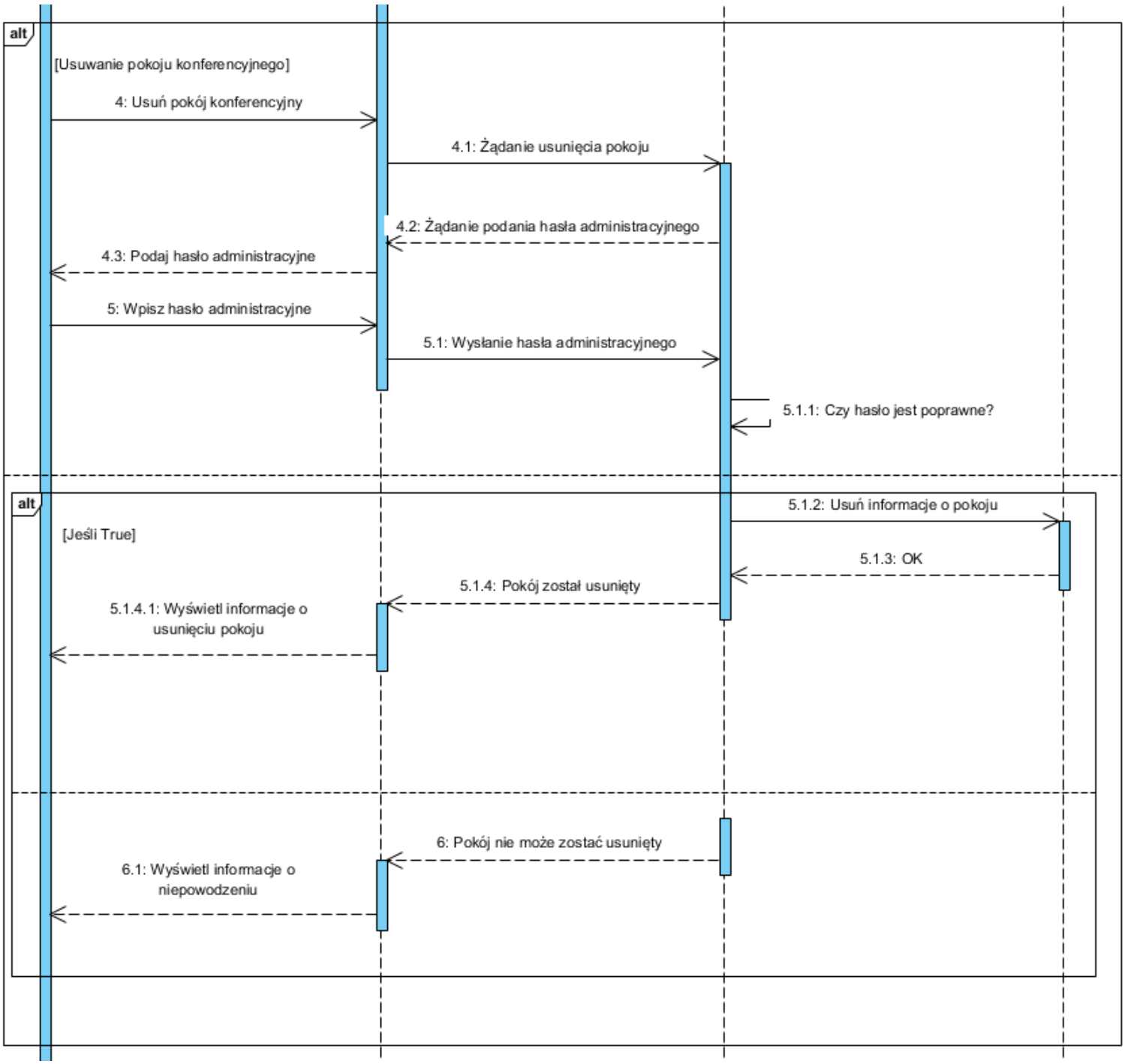
W niniejszym paragrafie zostanią przedstawione diagramy UML przedstawiające proces łączenia się użytkownika z serwerem, tworzenia pokoju konferencyjnego oraz usuwania pokoju konferencyjnego.



Rysunek 5: Diagram połączenia z serwerem



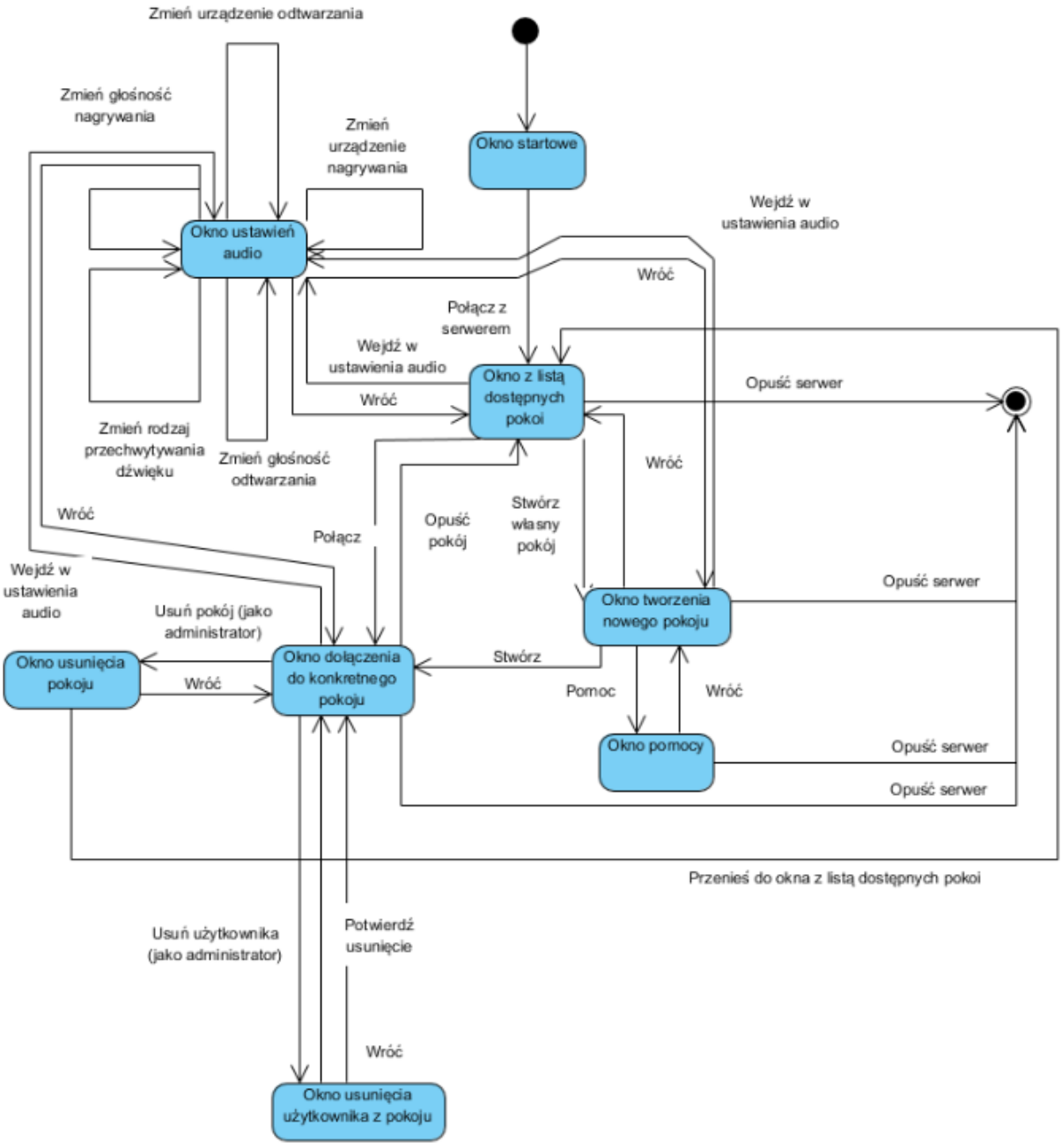
Rysunek 6: Diagram tworzenia pokoju konferencyjnego



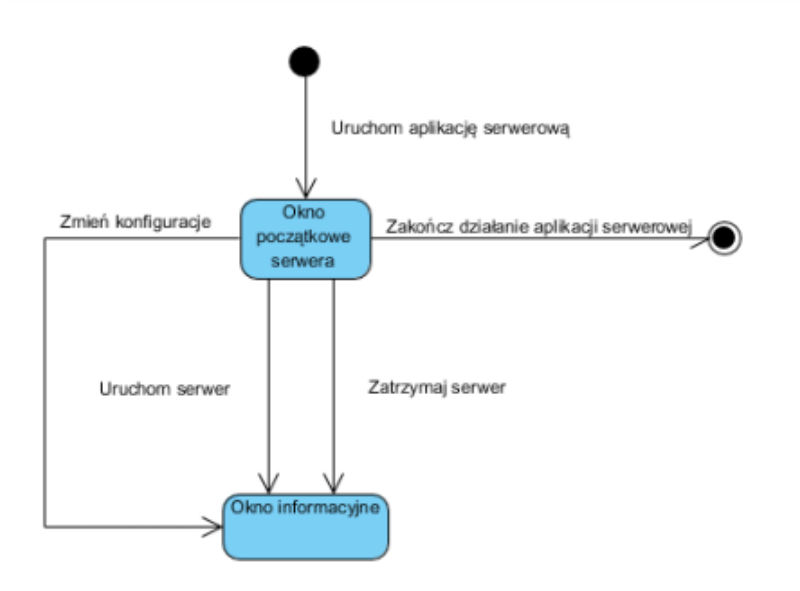
Rysunek 7: Diagram usuwania pokoju konferencyjnego

7.3 Diagramy stanów

W niniejszym paragrafie zostanią przedstawione diagramy UML przedstawiające przejścia stanów w aplikacji klienckiej oraz serwerowej.



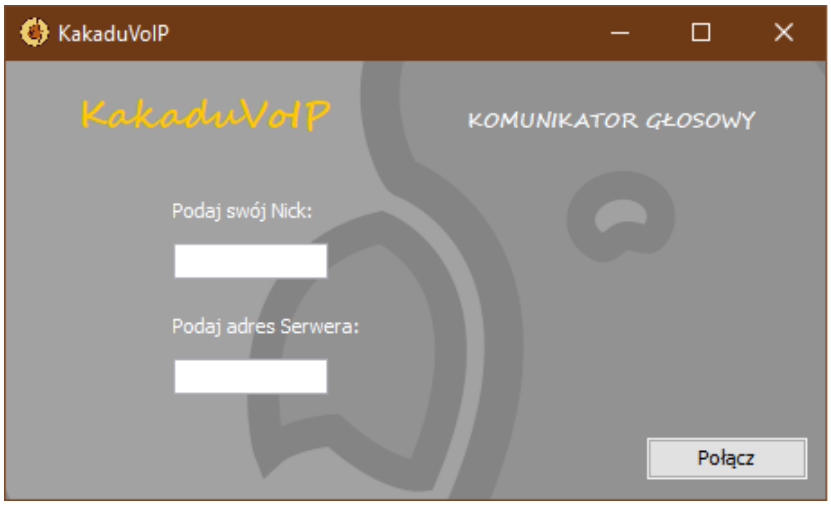
Rysunek 8: Diagram stanów w aplikacji klienckiej



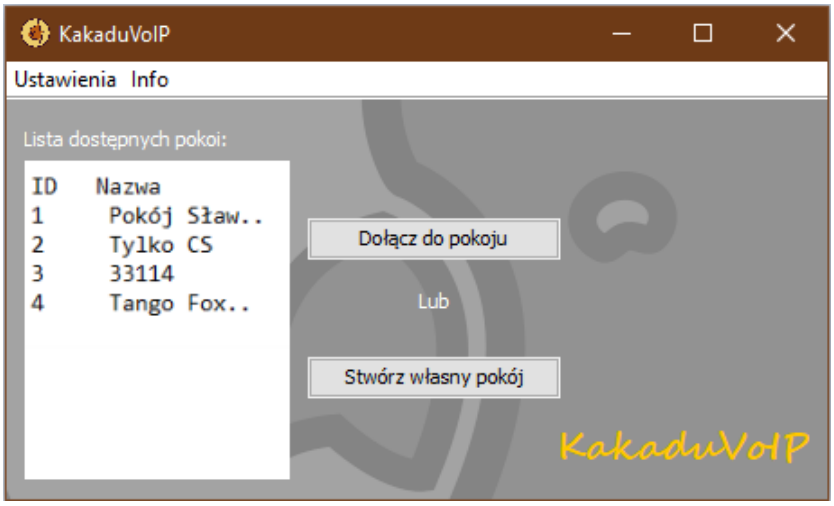
Rysunek 9: Diagram stanów w aplikacji serwerowej

8 Projekt interfejsu graficznego

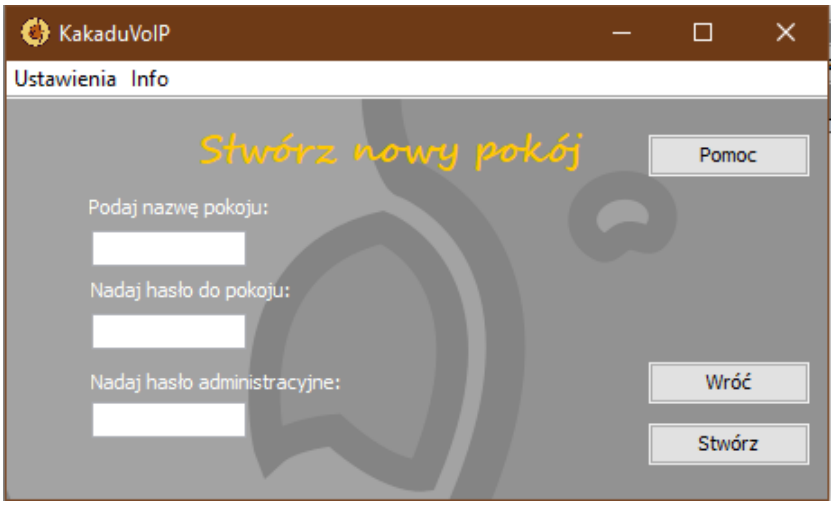
W paragrafie zostanie przedstawiony projekt interfejsu graficznego aplikacji.



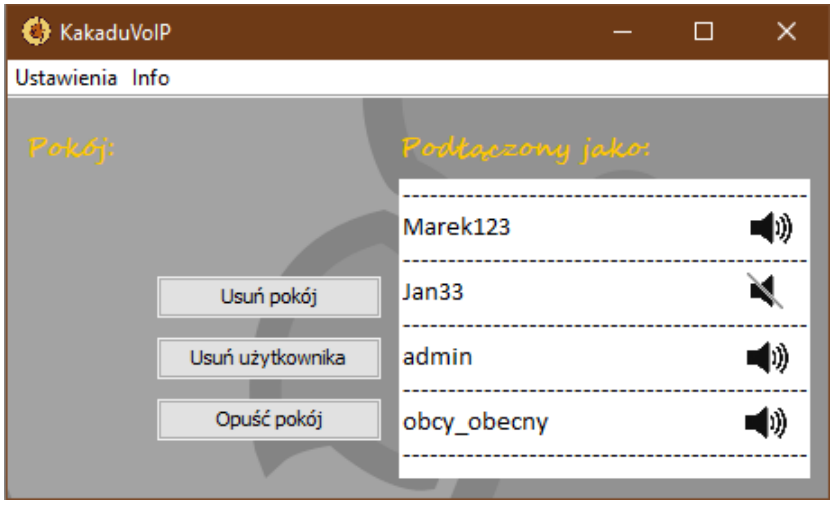
Rysunek 10: Interfejs ekranu połączenia



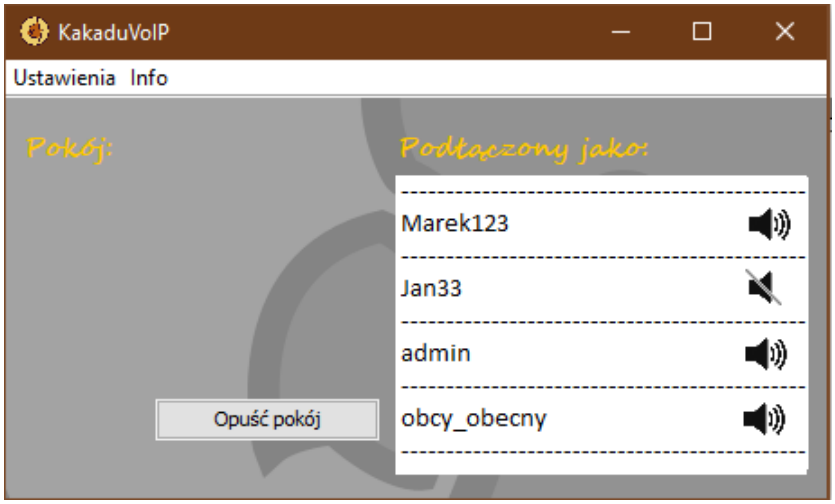
Rysunek 11: Interfejs ekranu głównego aplikacji



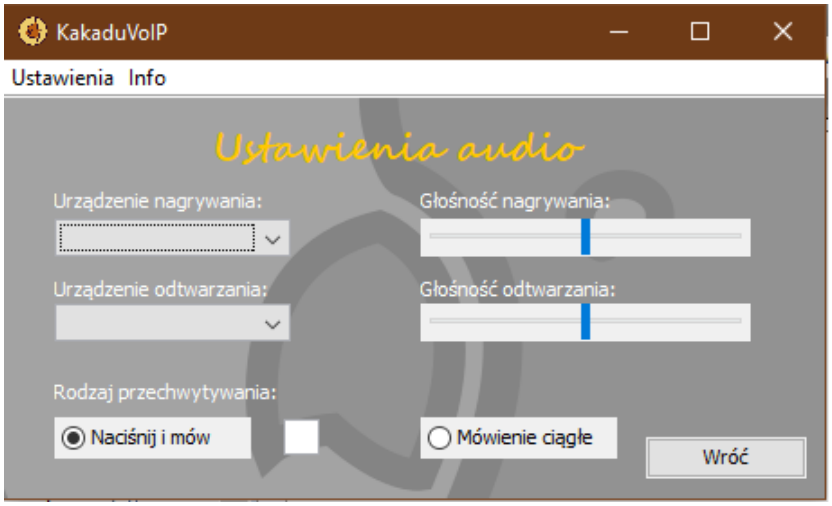
Rysunek 12: Interfejs ekranu tworzenia nowego pokoju



Rysunek 13: Interfejs ekranu pokoju podłączony jako administrator



Rysunek 14: Interfejs ekranu pokoju podłączony jako użytkownik



Rysunek 15: Interfejs ekranu ustawień audio

9 Najważniejsze metody i fragmenty kodu aplikacji

W tym paragrafie zostaną przedstawione najważniejsze fragmenty kodu aplikacji.

```java
public class Server {
    public static void main(String[] args) throws Exception {
        createTcpServer();
    }

    private static void createTcpServer() throws Exception {
        IoAcceptor acceptor = new NioSocketAcceptor();
        ResourceBundle serverProperties = ResourceBundle.getBundle("server");

        acceptor.getFilterChain().addLast( name: "logger", new LoggingFilter() );
        acceptor.getFilterChain().addLast( name: "codec", new ProtocolCodecFilter( new TextLineCodecFactory( Charset.forName( "UTF-8" ))));

        acceptor.setHandler(new TcpServerHandler());

        acceptor.getSessionConfig().setReadBufferSize( 2048 );
        acceptor.getSessionConfig().setIdleTime( IdleStatus.BOTH_IDLE,  i: 10 );

        int tcpPort = Integer.parseInt(serverProperties.getString( s: "server.tcp.port"));
        acceptor.bind(new InetSocketAddress(tcpPort));

    }
}
```

Rysunek 16: Inicjalizacja serwera TCP

```java
public class TcpServerHandler extends IoHandlerAdapter {
    @Override
    public void exceptionCaught(IoSession session, Throwable cause ) throws Exception
    {
        cause.printStackTrace();
    }

    @Override
    public void messageReceived( IoSession session, Object message ) throws Exception
    {
        String str = message.toString();
        if( str.trim().equalsIgnoreCase( s: "quit") ) {
            session.close();
            return;
        }

        Date date = new Date();
        session.write( date.toString() );
        System.out.println("Message written...");
    }

    @Override
    public void sessionIdle( IoSession session, IdleStatus status ) throws Exception
    {
        System.out.println( "IDLE " + session.getIdleCount( status ));
    }
}
```

Rysunek 17: Obsługa wiadomości przychodzących i wychodzących w serwerze

```java
public class MinaClientHandler extends IoHandlerAdapter
{
    private final Logger logger = (Logger) LoggerFactory.getLogger(getClass());
    private final String values;
    private boolean finished;

    public MinaClientHandler(String values)
    {
        this.values = values;
    }

    public boolean isFinished()
    {
        return finished;
    }

    @Override
    public void sessionOpened(IoSession session)
    {
        session.write(values);
    }

    @Override
    public void messageReceived(IoSession session, Object message)
    {
        System.out.println("Message received in the client..");
        System.out.println("Message is: " + message.toString());
    }

    @Override
    public void exceptionCaught(IoSession session, Throwable cause)
    {
        session.close();
    }
```

Rysunek 18: Obsługa wiadomości przychodzących i wychodzących u klienta

```java
public class TcpMessage {
    private static String HOSTNAME = "localhost";
    private static int PORT = 9123;

    public static void main(String[] args) throws IOException, InterruptedException
    {
        IoConnector connector = new NioSocketConnector();
        connector.getSessionConfig().setReadBufferSize(2048);

        connector.getFilterChain().addLast( name: "logger", new LoggingFilter());
        connector.getFilterChain().addLast( name: "codec", new ProtocolCodecFilter(new TextLineCodecFactory(Charset.forName("UTF-8"))));

        connector.setHandler(new MinaClientHandler( values: "Hello Server"));
        ConnectFuture future = connector.connect(new InetSocketAddress(HOSTNAME, PORT));
        future.awaitUninterruptibly();

        if (!future.isConnected())
        {
            return;
        }
        IoSession session = future.getSession();
        session.getConfig().setUseReadOperation(true);
        session.write( o: "Message to server");
        System.out.println(session.read().getMessage());

        System.out.println("After Writing");
        connector.dispose();

    }
```

Rysunek 19: Inicjalizacja klienta, wysłanie wiadomości i wyświetlenie odpowiedzi