Politechnika Poznańska

Wydział informatyki i telekomunikacji

Dokumentacja projektu

Dokumentacja projektu z zajęć Telefonia IP

Autorzy:

Adrian Golczak 136239
adrian.e.golczak@student.put.poznan.pl

Marcin Kubiak 136267
marcin.w.kubiak@student.put.poznan.pl

Wersja: v1.0.4

19.03.2020 r.

Spis treści

1 Charakterystyka ogólna projektu

Przedmiotem projektu pt. 'Opracowanie bezpiecznego systemu komunikacji głosowej w sieci IP (VoIP) wraz z jego implementacją' jest opracowanie aplikacji mobilnej na urządzenia z systemem Android wyposażonej w algorytmy RSA oraz AES-256 umożliwiające bezpieczną rozmowę pomiędzy dwoma użytkownikami aplikacji. Aplikacja będzie zaprojektowana tak, że osoba trzecia będzie w stanie podsłuchać tylko niewrażliwe dane, dzięki zastosowaniu RSA oraz AES-256. Główną koncepcją projektu jest stworzenie tzw. poczekalni, w której zalogowani użytkownicy bedą mogli się łączyć z kim tylko chcą i odbywać z nim rozmowę. W celu bezpieczeństwa, użytkownicy będą generować klucze publiczne oraz prywatne, które następnie będą używane do szyfrowania, deszyfrowania oraz przesyłaniu klucza szyfru blokowego AES, służącego do szyfrowania rozmowy. Aplikacja ma być łatwa w obsłudze oraz przejrzysta, dlatego będzie ograniczać się tylko do przyjmowania, odrzucania połączenia oraz rozmowy między dwoma użytkownikami. Dodatkowo w jednym pokoju jednocześnie będzie mogło przebywać dwóch klientów, a logowanie do poczekalni nie będzie wymagało podania hasła. Użytkownik będzie mógł kontrolować głośność za pomocą funkcji wbudowanych w telefonie.

2 Architektura systemu

System oparty jest o architekturę klient - serwer. Aplikacja kliencka zainstalowana została na urządzeniu mobilnym. Jej zadaniem jest udostępnienie interfejsu dla użytkownika w taki sposób, aby umożliwić sprawne i łatwe korzystanie z usług serwera. Aplikacja zapewnia również poziom bezpieczeństwa szyfrując/deszyfrując aplikację (serwer pełni rolę pośrednika, nie uczestniczy w komunikacji, nie jest w stanie "podsłuchaćżozmowy, gdyż ta jest szyfrowana.) W architekturze definiujemy 3 byty uczestniczące w procesie komunikacji:

- *Serwer* - aplikacja napisana w języku Java z użyciem frameworka Spring Boot. Jest instalowana na urządzeniu z wystarczającymi zasobami do zarządzania komunikacją (np. laptop, przy założeniu, że ilość żądań będzie nieznaczna),
- *Klient* - aplikacja na urządzenia mobilne z systemem Android,
- *Użytkownik* - osoba posiadająca zainstalowaną aplikację wraz z wylosowanym (przez serwer) unikalnym id,
- *Osoba akceptująca połączenie* - jest to użytkownik, który otrzymał informację poprzez interfejs, że inny użytkownik chce się z nim połączyć.

Użytkownik po uruchomieniu aplikacji klienckiej zostaje poproszony o wpisanie pseudonimu (pod warunkiem, że nie został już wpisany), następnie

klient wysyła żądanie do serwera o źarejestrowanie", żądanie składa się z wylosowanego uprzednio klucza publicznego, a także pseudonimu użytkownika. Serwer dodaje klienta do kolejki nadając mu unikalny identyfikator, od tej pory komunikacja oparta jest o ten identyfikator. W momencie w którym użytkownik zechce połączyć się innym użytkownikiem klient wysyła żądanie do serwera o sparowanie dwóch użytkowników (drugim jest osoba akceptująca połączenie). Jeśli połączenie zostanie zaakceptowane serwer tworzy sesję, a następnie wysyła odpowiednie komunikaty wraz z odpowiednimi kluczami publicznymi do klientów. Aplikacje mobilne używają kluczy publicznych do wylosowania 256 bitowego klucza AES po 128 bitów każdy, gdzie pierwsze 128 bitów należy do osoby żądającej połączenia, a ostatnie 128 bitów należy do osoby akceptującej. Obie strony wymieniają się zaszyfrowanymi częściami klucza, ostatnim krokiem jest potwierdzenie otrzymania części drugiej strony. Od tego momentu komunikacja jest szyfrowana AES-256, aż do zakończenia rozmowy.

3 Wymagania

Poniżej opisane zostaną wymagania funkcjonalne i niefunkcjonalne aplikacji, z wyróżnieniem różnych stanów użytkownika.

3.1 Funkcjonalne

Użytkownik niezalogowany:

- podanie pseudonimu,
- logowanie się do poczekalni.

Użytkownik zalogowany:

- wysyłanie próśb o połączenie,
- akceptacja prośby od drugiego użytkownika,
- generowanie klucza publicznego i prywatnego wykorzystując algorytm RSA,
- odrzucenie prośby o połączenie,
- opuszczenie poczekalni oraz trwającej rozmowy,
- wgląd do listy aktywnych użytkowników z podziałem na osoby rozmawiające i czekające na rozmowę.

3.2 Niefunkcjonalne

- łączenie dwóch użytkowników,
- generowanie ID sesji

- negocjacje klucza o rozmiarze 256 bitów na potrzeby AES-256,
- szyfrowanie rozmowy wykorzystując AES-256,
- minimalna wersja systemu Android: 10.0.0,
- brak wymogu podania hasła przez użytkownika przy logowaniu,
- limit użytkowników w poczekalni podyktowany mocą obliczeniową serwera,
- jedno urządzenie mobilne to jedna instancja klienta,
- w ramach jednego pokoju może przebywać jedynie dwóch użytkowników,
- zmiana głośności rozmowy za pomocą wbudowanych funckji sprzętowych urządzenia.

4 Technologie, narzędzia, środowisko, biblioteki, kodeki

W procesie tworzenia systemu zostaną wykorzystane technologie i narzędzia umożliwiające komunikację pomiędzy klientami, ułatwiające pisanie dokumentacji, a także upraszczające proces pisania kodu źródłowego. Użyte zostaną między innymi:

- *TeXstudio*
- *InteliJ*
- *Java11*
- *SpringBoot*
- *AndroidStudio*
- *javax.sound*
- *jcodec*

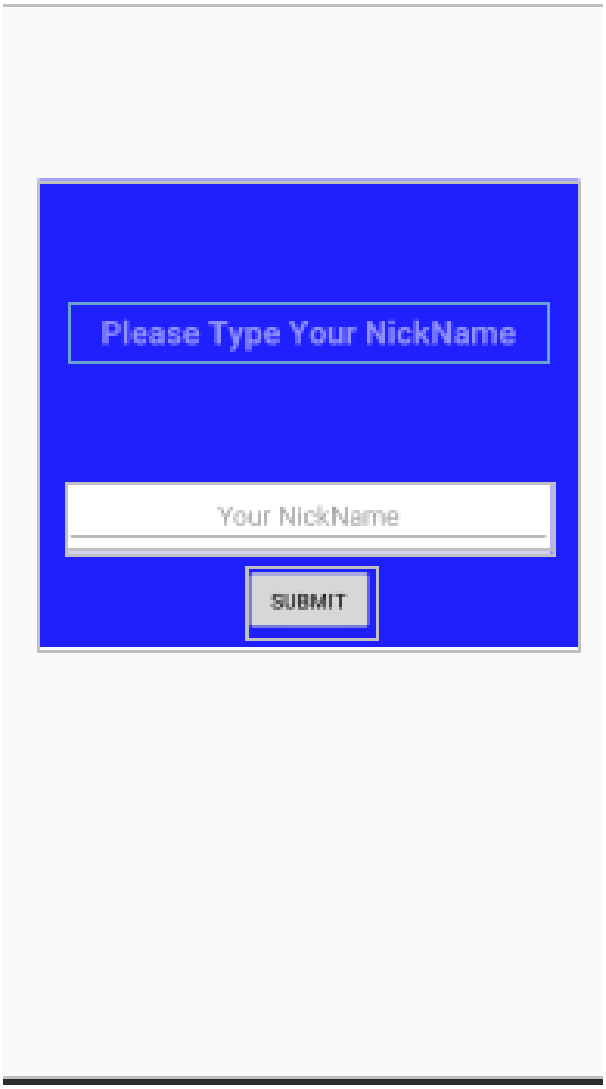
5 Opis najważniejszych protokołów

W poniższym rozdziale przedstawimy najważniejsze, wg nas, protokoły składające się na naszą aplikację.

5.1 Protokół łączenia użytkowników

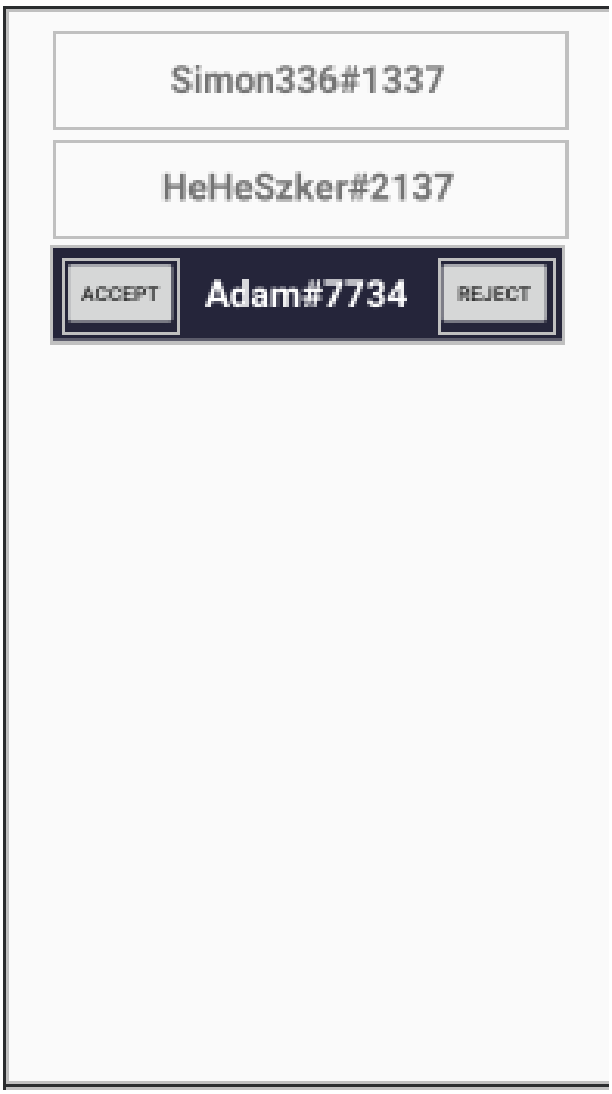
6 Ekrany

Poniżej zaprezentowano wstępne makiety ekranów jakie będą używane w aplikacji mobilnej, do obsługi dodawania użytkownika do kolejki, dzwonienia, odbierania połączeń, rozmowy i kończenia rozmowy. Wszystkie ekrany są jedynie koncepcją interfejsu i nie należy traktować ich jako produktu końcowego.

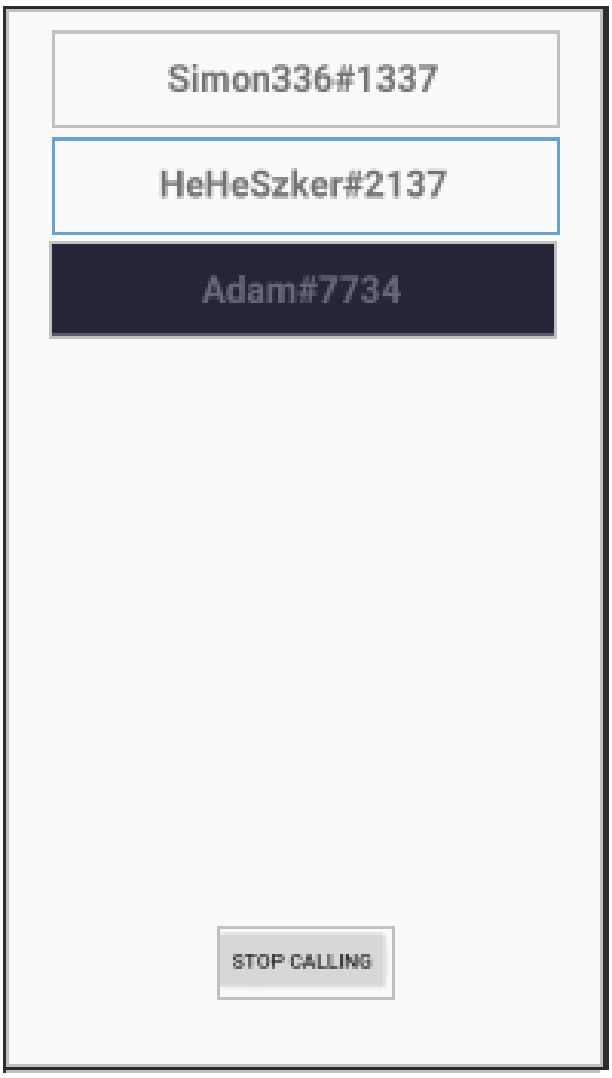


Rysunek 1: Pierwszy ekran z prośbą o podanie pseudonimu użytkownika.

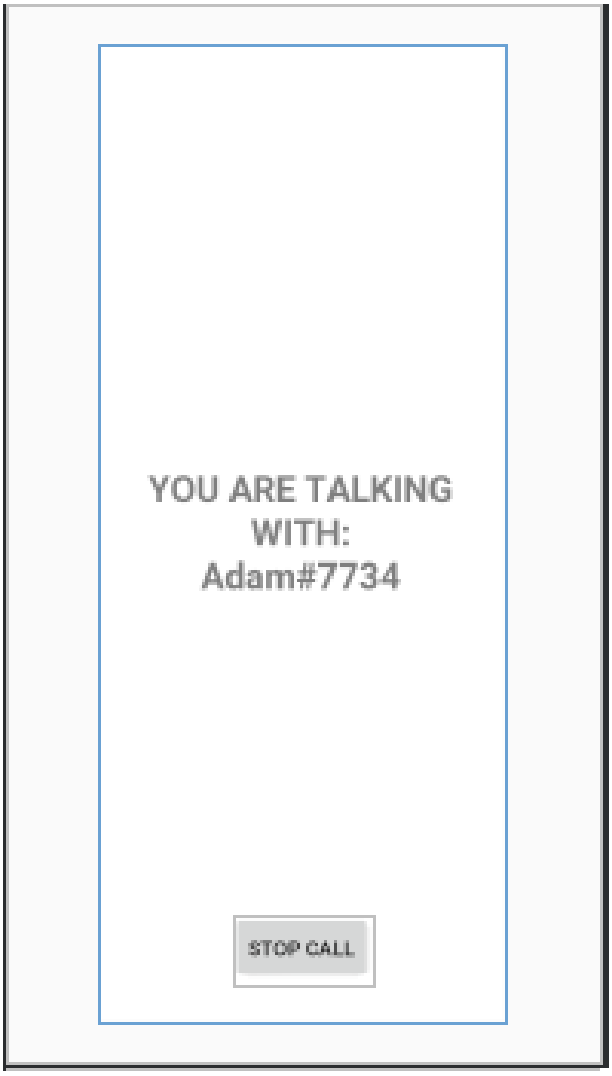
Rysunek 2: Ekran z widoczną ”poczekalnią”, w której znajduje się trzech użytkowników.



Rysunek 3: Informacja o nadchodzącym połączenia od użytkownika Adam.



Rysunek 4: Próba połączenia się z użytkownikiem Adam.

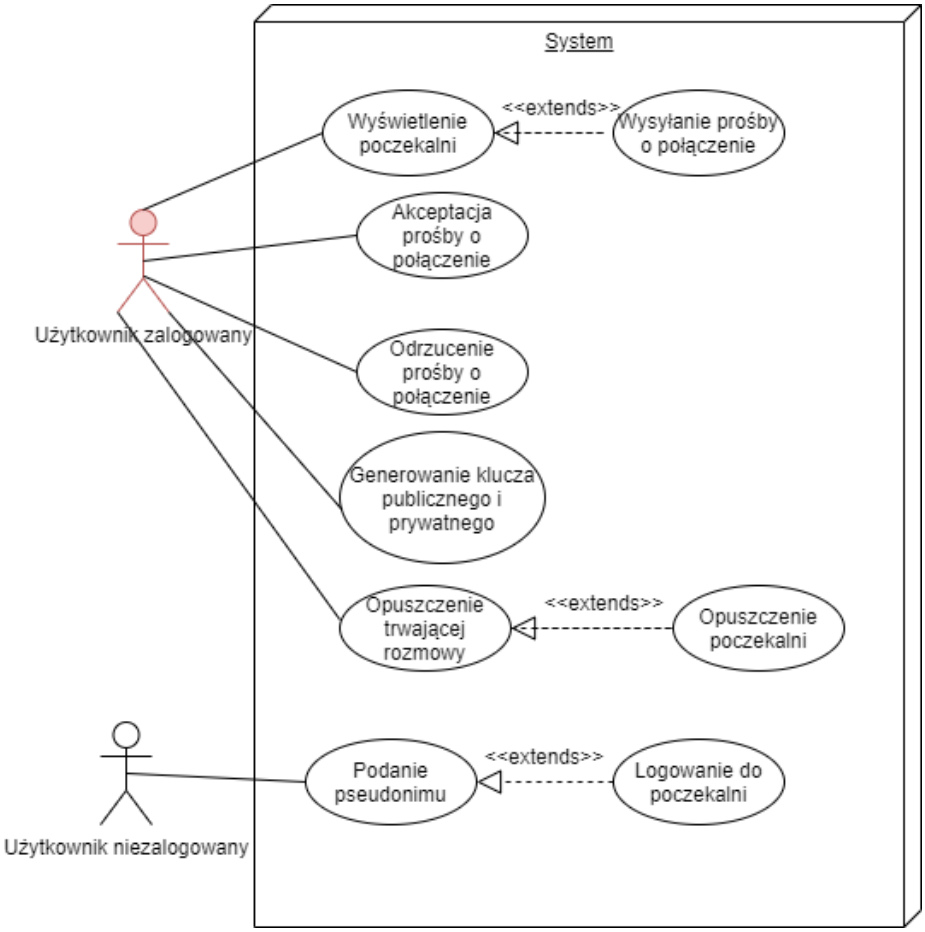


Rysunek 5: Rozmowa z użytkownikiem Adam.

7 Diagramy UML

W tym rozdziale skupimy się na 4 typach diagramów UML: przypadków użycia, przebiegów, stanów oraz klas.

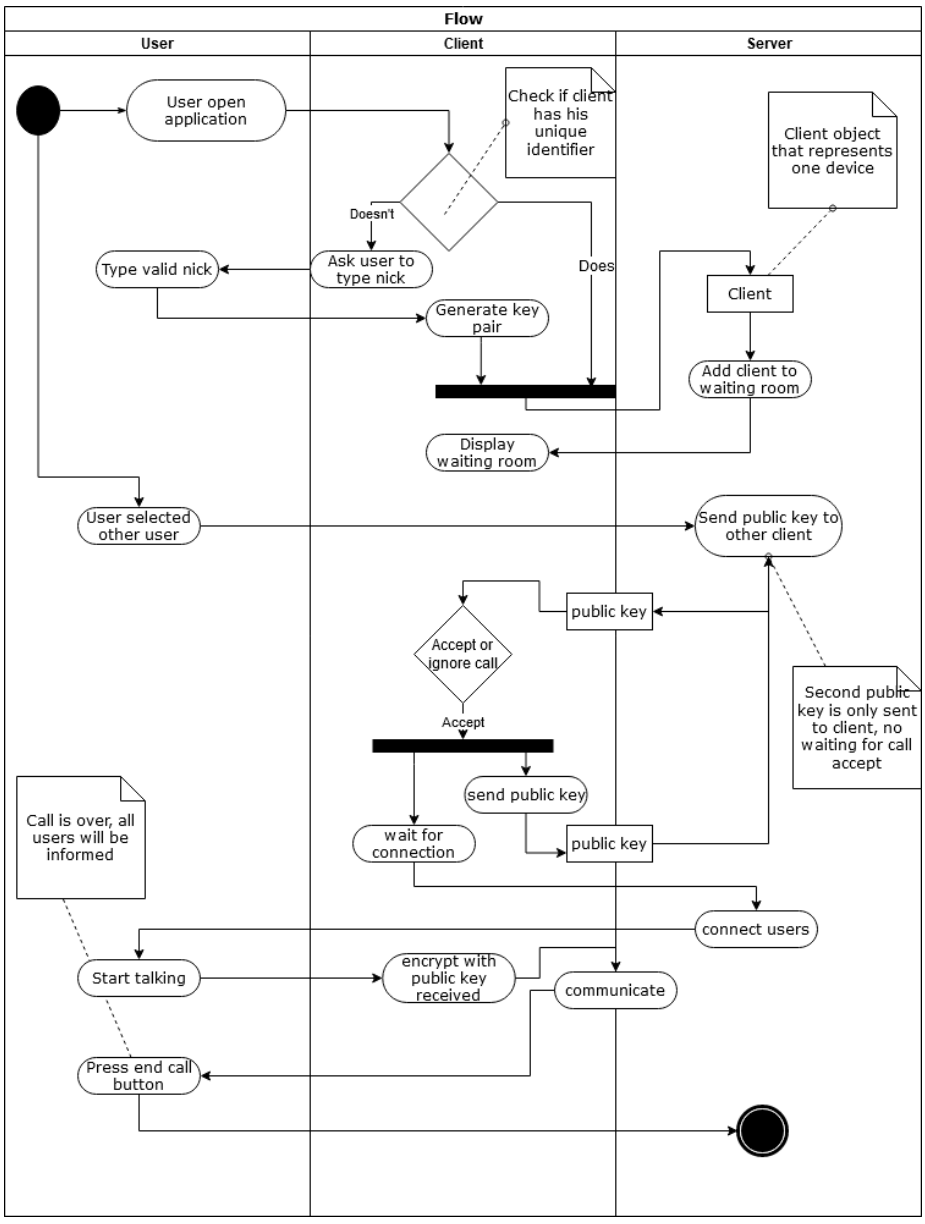
7.1 Diagram przypadków użycia



Rysunek 6: Diagram przypadków użycia z podziałem na dwóch aktórów: użytkownika zalogowanego i niezalogowanego

7.2 Uproszczony diagram przepływu

Poniżej zaprezentowano uproszczony diagram przepływu. Diagram podzielono na 3 obiekty: użytkownika aplikacji, klienta, czyli aplikację mobilną będącą pośrednikiem pomiędzy użytkownikiem, a serwerem, i serwer który obsługuje wszystkie zapytania i odpowiednio je przetwarza, zarządza sesjami i połączeniami.



Rysunek 7: Uproszczony diagram przepływu przedstawiajacy inicjalizację sesji, a także dodania użytkownika do poczekalni.

8 Implementacja serwera

W tej sekcji prezentujemy najciekawsze fragmenty kodu aplikacji serwerowej. Jako wzorzec projektowy do wytwarzania oprogramowania użyliśmy Domain Driven Design, dzięki czemu podzieliliśmy kod na domeny, a to pozwoliło utrzymać go w "czystości" i ułatwia ewentualną późniejszą rozbudowę.

8.1 RegisterService

Serwis ten ma za zadanie zarządzać poczekalnią, dodawać i usuwać z niej użytkowników. Poczekalnia jest wstrzykniętą listą użytkowników przy użyciu Inversion of Controll Container'a, jako implementacji Dependency Injection.

```
@Service
public class BasicUserRegisterService implements UserRegisterService {
    private AtomicInteger seqNumber = new AtomicInteger();
    @Resource(name = "waitingRoom")
    private List<VoIPUser> users;

    @Override
    public RegisterResponse registerUser(RegisterRequest request) {

        String newNick = generateNickFromRequest(request.getNick());
        String userToken = UUID.randomUUID()
                                .toString();
        String publicKey = request.getPublicKey();
        return addToWaitingRoom(newNick, userToken, publicKey);
    }

    private RegisterResponse addToWaitingRoom(String newNick, String userToken, String publicKey)
        VoIPUser user = new VoIPUser();
        user.setNick(newNick);
        user.setUserToken(userToken);
        user.setReadyToTalk(true);
        user.setPublicKey(publicKey);
        users.add(user);
        return new RegisterResponse(newNick, userToken);
    }

    private String generateNickFromRequest(String nick) {
        String s = nick + " " + seqNumber;
        seqNumber.getAndIncrement();
        return s;
    }
}
```

Rysunek 8: Kody serwisu zarządzającego użytkownikami i poczekalnią.

8.2 ConnectionService

Serwis odpowiedzialny za łączenie i rozłączanie użytkowników pośrednio wykorzystuje SessionService który zarządza dodatkowo sesjami użytkowników.

```java
@Service
@RequiredArgsConstructor
public class BasicUserConnectionService implements UserConnectionService {

    private static final String OK = "OK";
    private static final String BUSY = "BUSY";

    @Resource(name = "waitingRoom")
    private List<VoIPUser> users;

    private final CommunicationForwarderService forwarderService;
    private final UserSessionService sessionService;

    @Override
    public ConnectionResponse connect(String respondersSessionToken, String sessionIdToken) {
        sessionService.addUserToSession(respondersSessionToken, sessionIdToken);

        return new ConnectionResponse(OK);
    }

    @Override
    public ConnectionResponse refuse(String respondersSessionToken, String sessionIdToken) {
        String initiatorsToken = sessionService.killSession(respondersSessionToken, sessionIdToken);
        forwarderService.sendRefuseMessage(initiatorsToken);

        return new ConnectionResponse(OK);
    }

    @Override
    public ConnectionResponse tryConnectWith(String initiatorsToken, String responderNick) {
        if (!userIsCurrentlyOnCall(responderNick)) {
            String sessionIdToken = sessionService.createNewSession(initiatorsToken);
            forwarderService.makeCall(initiatorsToken, responderNick, sessionIdToken);
            return new ConnectionResponse(OK);
        }
        return new ConnectionResponse(BUSY);
    }
```

Rysunek 9: Klasa zarządzająca połączeniem użytkowników podczas próby rozmowy.