Android Audio Chat

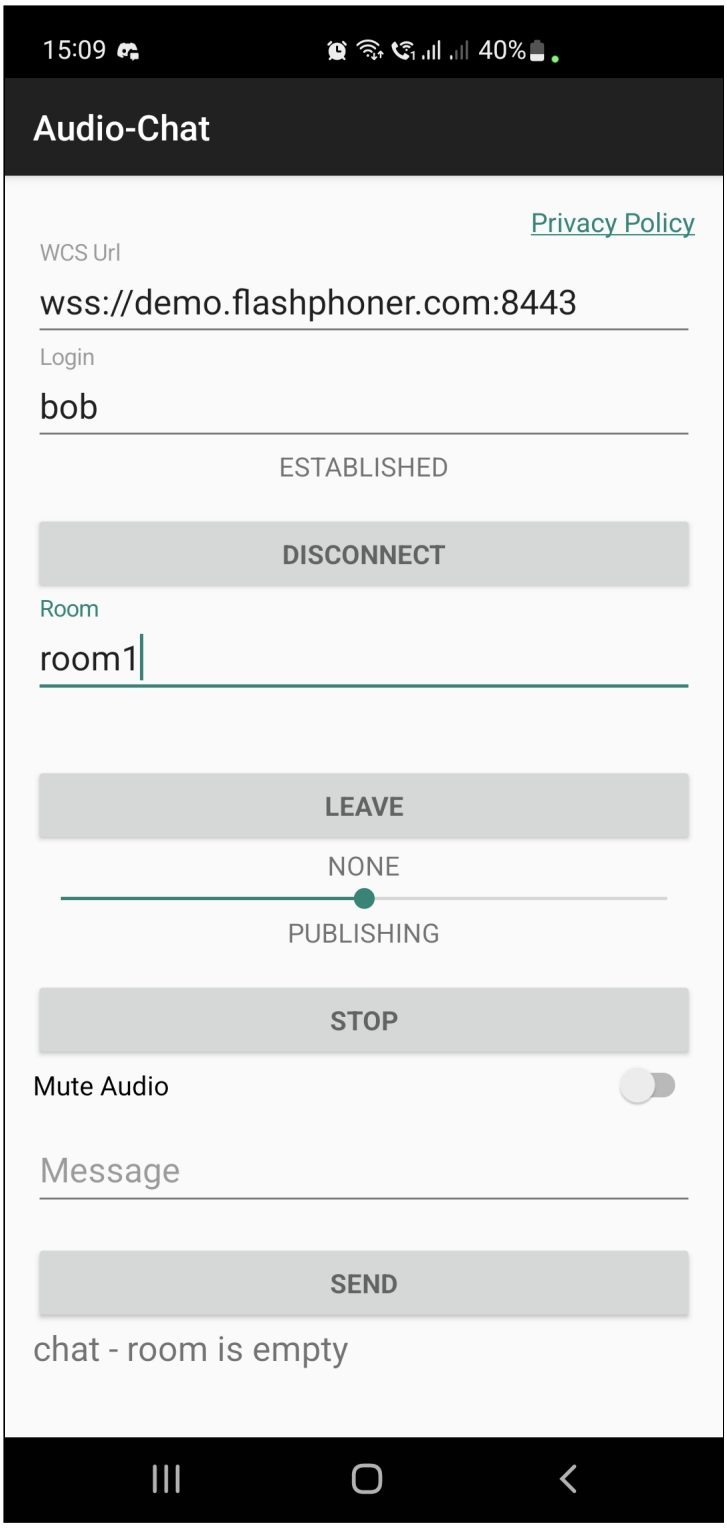
Описание

Данный пример может использоваться для аудиочата между двумя участниками и позволяет публиковать и проигрывать аудио WebRTC-поток с Web Call Server .

Ниже представлен пример аудиочата в приложении

Поля ввода приложения:

- `WCS URL`, где `wss://demo.flashphoner.com:8443/` - адрес WCS-сервера
- `Login`, где `bob` - имя участника
- `Room`, где `room1` - имя "комнаты" чата



Работа с кодом примера

Для разбора кода возьмем класс AudioChatActivity.java примера audio chat, который доступен для скачивания в соответствующей сборке Android SDK 1.1.0.61.

В отличие от прямого подключения к серверу методом `createSession()`, для управления подключением к серверу и конференции используется объект `RoomManager`. Соединение с сервером устанавливается при создании объекта `RoomManager`, а для присоединения к конференции вызывается метод `RoomManager.join()`, При присоединении к новой "комнате" методом `RoomManager.join()`, создается объект `Room` для работы с этой "комнатой". Для работы с участниками конференции используются объекты `Participant`. Все события, происходящие в "комнате" (присоединение/выход пользователя, отправленные сообщения), транслируются другим участникам, подключенным к этой "комнате".

Например, в следующем коде подключаемся к "комнате" и запрашиваем список других участников:

```
room = roomManager.join(roomOptions);
room.on(new RoomEvent() {
    public void onState(final Room room) {
        for (final Participant participant : room.getParticipants()) {
            ...
        }
        ...
    }
    ...
});
```

1. Инициализация API

`Flashphoner.init()` code

```
Flashphoner.init(this);
```

2. Создание объекта для рендера полученного аудио потока

`SurfaceViewRenderer` code

```
renderer = new SurfaceViewRenderer(this);
```

3. Подключение к серверу

`Flashphoner.createRoomManager()` code

Методу передается объект `RoomManagerOptions` со следующими параметрами:

- URL WCS-сервера
- имя пользователя для присоединения к чат-комнате

```
RoomManagerOptions roomManagerOptions = new
RoomManagerOptions(mWcsUrlView.getText().toString(),
mLoginView.getText().toString());

/**
 * RoomManager object is created with method createRoomManager().
 * Connection session is created when RoomManager object is created.
 */
roomManager = Flashphoner.createRoomManager(roomManagerOptions);
```

4. Получение от сервера события, подтверждающего успешное соединение

`RoomManager.onConnected()` code

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mConnectButton.setText(R.string.action_disconnect);
            mConnectButton.setTag(R.string.action_disconnect);
            mConnectButton.setEnabled(true);
            mConnectStatus.setText(connection.getStatus());
            mJoinButton.setEnabled(true);
        }
    });
}
```

5. Присоединение к конференции

`RoomManager.join()` code

Методу передается объект `RoomOptions` с именем комнаты конференции

```
RoomOptions roomOptions = new RoomOptions();
roomOptions.setName(mJoinRoomView.getText().toString());

/**
 * The participant joins a audio chat room with method RoomManager.join().
 * RoomOptions object is passed to the method.
 * Room object is created and returned by the method.
 */
room = roomManager.join(roomOptions);
```

6. Получение от сервера события, описывающего текущее состояние комнаты

`Room.onState()` code

При получении данного события количество и состав других участников определяется с помощью метода `Room.getParticipants()`. Если количество участников более 2, текущий участник выходит из комнаты.

Если текущий участник остается в комнате, запускается проигрывание потока от другого участника при помощи метода `Participant.play()`

```
if (room.getParticipants().size() >= 2) {
    room.leave(null);
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                mJoinStatus.setText("Room is full");
                mJoinButton.setEnabled(true);
            }
        }
    );
    return;
}


final StringBuffer chatState = new StringBuffer("participants: ");

/**
 * Iterating through the collection of the other participants returned by
method Room.getParticipants().
 * There is corresponding Participant object for each participant.
 */
for (final Participant participant : room.getParticipants()) {
    /**
     * A player view is assigned to each of the other participants in the
room.
     */
    chatState.append(participant.getName()).append(",");
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                participant.play(renderer);
                mParticipantName.setText(participant.getName());
            }
        }
    );
}
```

7. Публикация аудиопотока

`Room.publish()` code

Методу передаются ограничения, указывающие, что публиковаться должно только аудио

```
case PUBLISH_REQUEST_CODE: {
    if (grantResults.length == 0 ||
            grantResults[0] != PackageManager.PERMISSION_GRANTED) {

        Log.i(TAG, "Permission has been denied by user");
    } else {
        /**
         * Stream is created and published with method Room.publish().
         */
        StreamOptions streamOptions = new StreamOptions();
        streamOptions.setConstraints(new Constraints(true, false));
        stream = room.publish(null, streamOptions);
        ...
        Log.i(TAG, "Permission has been granted by user");
    }
}
```

8. Получение от сервера события, сигнализирующего о присоединении к конференции другого участника

`Room.onJoined()` code

```
@Override
public void onJoined(final Participant participant) {
    /**
     * When a new participant joins the room, a player view is assigned to
that participant.
     */
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                mParticipantName.setText(participant.getName());
                addMessageHistory(participant.getName(), "joined");
            }
        }
    );
}
```

9. Получение от сервера события, сигнализирующего о публикации видеопотока другим участником

`Room.onPublished()` code

При получении данного события поток, опубликованный участником, воспроизводится с помощью метода `Participant.play()`. Этому методу передается объект `SurfaceViewRenderer`, в котором будет проигрываться аудио

```
@Override
public void onPublished(final Participant participant) {
    /**
```

```
     * When one of the other participants starts publishing, playback of the
stream published by that participant is started.
     */
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                participant.play(renderer);
            }
        }
    );
}
```

10. Получение от сервера события, сигнализирующего об отправке сообщения другим участником

`Room.onMessage()` code

```
@Override
public void onMessage(final Message message) {
    /**
     * When one of the participants sends a text message, the received
message is added to the messages log.
     */
    runOnUiThread(
        new Runnable() {
            @Override
            public void run() {
                addMessageHistory(message.getFrom(), message.getText());
            }
        }
    );
}
```

11. Отправка сообщения другому участнику

`Participant.sendMessage()` code

```
mSendButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        String text = mMessage.getText().toString();
        if (!"".equals(text)) {
            for (Participant participant : room.getParticipants()) {
                participant.sendMessage(text);
            }
            addMessageHistory(mLoginView.getText().toString(), text);
            mMessage.setText("");
        }
    }
});
```

12. Остановка публикации аудиопотока при нажатии Unpublish

`Room.unpublish()` code

```
@Override
public void onClick(View view) {
    if (mPublishButton.getTag() == null ||
Integer.valueOf(R.string.action_publish).equals(mPublishButton.getTag())) {
        ActivityCompat.requestPermissions(AudioChatActivity.this,
                new String[]{Manifest.permission.RECORD_AUDIO},
                PUBLISH_REQUEST_CODE);
    } else {
        mPublishButton.setEnabled(false);
        /**
         * Stream is unpublished with method Room.unpublish().
         */
        room.unpublish();
    }
    ...
}
```

13. Выход из комнаты при нажатии Leave

`Room.leave()` code

Методу передается обработчик ответа REST хука, используемого при выходе из комнаты

```
room.leave(new RestAppCommunicator.Handler() {
    @Override
    public void onAccepted(Data data) {
        runOnUiThread(action);
    }

    @Override
    public void onRejected(Data data) {
        runOnUiThread(action);
    }
});
```

14. Закрытие соединения

`RoomManager.disconnect()` code

```
mConnectButton.setEnabled(false);

/**
 * Connection to WCS server is closed with method RoomManager.disconnect().
 */
roomManager.disconnect();
```

15. Включение/выключение аудио для публикуемого потока

`Stream.unmuteAudio()`, `Stream.muteAudio()` code

```
mMuteAudio = (Switch) findViewById(R.id.mute_audio);
mMuteAudio.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            stream.muteAudio();
        } else {
            stream.unmuteAudio();
        }
    }
});
```