aws

Panduan Pengguna Obrolan

Amazon IVS

Amazon IVS: Panduan Pengguna Obrolan

Table of Contents

Apa itu IVS Obrolan Amazon?

IVSObrolan Amazon adalah fitur obrolan langsung yang dikelola untuk mengikuti streaming video langsung. Dokumentasi dapat diakses dari halaman arahan IVS dokumentasi Amazon, di bagian IVS Obrolan Amazon:

- Panduan Pengguna Obrolan — Dokumen ini, beserta semua halaman Panduan Pengguna lainnya yang tercantum di panel navigasi.
- APIReferensi Obrolan — Control-plane API ()HTTPS.
- APIReferensi Pesan Obrolan — Data-plane API ()WebSocket.
- SDKReferensi untuk klien obrolan: Android, iOS, dan JavaScript.

Memulai dengan IVS Obrolan Amazon

Amazon Interactive Video Service (IVS) Chat adalah fitur obrolan langsung yang dikelola untuk mengikuti streaming video langsung Anda. (IVSObrolan juga dapat digunakan tanpa streaming video.) Anda dapat membuat ruang obrolan dan mengaktifkan sesi obrolan di antara pengguna Anda.

IVSObrolan Amazon memungkinkan Anda fokus untuk membangun pengalaman obrolan yang disesuaikan bersama video langsung. Anda tidak perlu mengelola infrastruktur atau mengembangkan dan mengonfigurasi komponen alur kerja obrolan. IVSObrolan Amazon dapat diskalakan, aman, andal, dan hemat biaya.

IVSObrolan Amazon berfungsi paling baik untuk memfasilitasi pengiriman pesan antara peserta streaming video langsung dengan awal dan akhir.

Sisa dokumen ini membawa Anda melalui langkah-langkah untuk membangun aplikasi obrolan pertama Anda menggunakan IVS Obrolan Amazon.

Contoh: Aplikasi demo berikut tersedia (tiga contoh aplikasi klien dan aplikasi server backend untuk pembuatan token):

- Demo Web IVS Obrolan Amazon
- IVSObrolan Amazon untuk Demo Android
- IVSObrolan Amazon untuk Demo iOS
- Backend Demo IVS Obrolan Amazon

Penting: Ruang obrolan yang tidak memiliki koneksi atau pembaruan baru selama 24 bulan akan dihapus secara otomatis.

Topik

Langkah 1: Jalankan Pengaturan Awal

Sebelum melanjutkan, Anda harus:

1. Buat akun AWS.
2. Menyiapkan pengguna root dan administratif.
3. Mengatur izin AWS IAM (Identity and Access Management). Menggunakan kebijakan yang ditentukan di bawah ini.

Untuk langkah-langkah spesifik untuk semua hal di atas, lihat Memulai Streaming IVS Latensi Rendah di IVSPanduan Pengguna Amazon. Penting: Di “Langkah 3: Siapkan IAM Izin,” gunakan kebijakan ini untuk IVS Obrolan:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ivschat:CreateChatToken",
            "ivschat:CreateLoggingConfiguration",
            "ivschat:CreateRoom",
            "ivschat:DeleteLoggingConfiguration",
            "ivschat:DeleteMessage",
            "ivschat:DeleteRoom",
            "ivschat:DisconnectUser",
            "ivschat:GetLoggingConfiguration",
            "ivschat:GetRoom",
            "ivschat:ListLoggingConfigurations",
            "ivschat:ListRooms",
            "ivschat:ListTagsForResource",
            "ivschat:SendEvent",
            "ivschat:TagResource",
            "ivschat:UntagResource",
            "ivschat:UpdateLoggingConfiguration",
            "ivschat:UpdateRoom"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
```

```
         "Action": [
            "servicequotas:ListServiceQuotas",
            "servicequotas:ListServices",
            "servicequotas:ListAWSDefaultServiceQuotas",
            "servicequotas:ListRequestedServiceQuotaChangeHistoryByQuota",
            "servicequotas:ListTagsForResource",
            "cloudwatch:GetMetricData",
            "cloudwatch:DescribeAlarms"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogDelivery",
            "logs:GetLogDelivery",
            "logs:UpdateLogDelivery",
            "logs:DeleteLogDelivery",
            "logs:ListLogDeliveries",
            "logs:PutResourcePolicy",
            "logs:DescribeResourcePolicies",
            "logs:DescribeLogGroups",
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy",
            "iam:CreateServiceLinkedRole",
            "firehose:TagDeliveryStream"
         ],
         "Resource": "*"
      }
   ]
}
```

Langkah 2: Buat Ruang Obrolan

Ruang IVS obrolan Amazon memiliki informasi konfigurasi yang terkait dengannya (misalnya, panjang pesan maksimum).

Petunjuk di bagian ini menunjukkan cara menggunakan konsol atau AWS CLI mengatur ruang obrolan (termasuk pengaturan opsional untuk meninjau pesan dan/atau mencatat pesan) dan membuat ruangan.

Petunjuk Konsol untuk Membuat Ruang IVS Obrolan

Langkah-langkah ini dibagi menjadi sejumlah fase, dimulai dengan pengaturan ruang awal dan diakhiri dengan pembuatan ruang akhir.

Atau, Anda dapat mengatur ruang sehingga pesan dapat ditinjau. Misalnya, Anda dapat memperbarui konten atau metadata pesan, menolak pesan agar tidak terkirim, atau membiarkan pesan asli tetap tersampaikan. Hal tersebut tercakup dalam Menyiapkan Tinjauan Pesan Ruang (Opsional).

Atau, Anda dapat mengatur ruang agar pesan dapat dibuatkan log. Misalnya, jika Anda memiliki pesan yang dikirim ke ruang obrolan, Anda dapat mencatatnya ke bucket Amazon S3, Amazon CloudWatch, atau Amazon Kinesis Data Firehose. Hal tersebut tercakup dalam Menyiapkan Log Pesan (Opsional).

Pengaturan Ruang Awal

1. Buka konsol IVS Obrolan Amazon.

   (Anda juga dapat mengakses IVS konsol Amazon melalui Konsol AWS Manajemen.)
2. Dari bilah navigasi, gunakan menu tarik-turun Pilih Wilayah untuk memilih wilayah. Ruang baru Anda akan dibuat di wilayah ini.
3. Di kotak Memulai (kanan atas), pilih Ruang IVS Obrolan Amazon. Jendela Buat ruang akan muncul.

Create room Info

Rooms are the central Amazon IVS Chat resource. Clients can connect to a room to exchange messages with other clients who are connected to the room. Rooms that are inactive for 24 months will be automatically deleted. Learn more

How Amazon IVS Chat works

Setup

Room name – *optional*

name

Maximum length: 128 characters. May include numbers, letters, underscores (_), and hyphens (-).

Room configuration

- Default configuration
  Use the default maximum value of message limits
- Custom configuration
  Specify your own chat message limits

Message character limit Info

500 characters per message

Maximum message rate Info

10 messages per second

Message review handler Info

Review messages before they are sent to the room

- Disabled
  Messages will not be reviewed
- Handle with AWS Lambda
  Create or select an AWS Lambda function

Message logging Info

Automatically log chat messages

When enabled, messages from the chat room are logged automatically. Logged content can be managed directly in the destination services.

- Disabled
  Chat messages will not be logged
- Automatically log messages and events
  Create or select logging configurations

4. Di bagian Pengaturan, secara opsional tentukan Nama ruang. Nama kamar tidak unik, tetapi mereka menyediakan cara bagi Anda untuk membedakan kamar selain ruangan ARN (Nama Sumber Daya Amazon).
5. Di bagian Pengaturan > Konfigurasi ruang, terima Konfigurasi default, atau pilih Konfigurasi kustom, kemudian konfigurasikan Panjang pesan maksimum dan/atau Kecepatan pesan maksimum.
6. Jika Anda ingin meninjau pesan, lanjutkan dengan Menyiapkan Tinjauan Pesan Ruang (Opsional) di bawah. Jika tidak, lewati proses tersebut (yaitu, terima Handler Tinjauan Pesan > Nonaktif) dan lanjutkan langsung ke Pembuatan Ruang Akhir.

Menyiapkan Tinjauan Pesan Ruang (Opsional)

1. Di bawah Message Review Handler, pilih Tangani dengan AWS Lambda. Bagian Handler Tinjauan Pesan diperluas untuk menampilkan opsi tambahan.
2. Konfigurasikan Hasil fallback menjadi Izinkan atau Tolak pesan jika handler tidak mengembalikan respons yang valid, menemui kesalahan, atau melebihi periode batas waktu.
3. Tentukan Fungsi Lambda yang sudah ada atau gunakan Buat fungsi Lambda untuk membuat fungsi baru.

   Fungsi lambda harus berada di AWS akun yang sama dan AWS wilayah yang sama dengan ruang obrolan. Anda harus memberikan izin SDK layanan Obrolan Amazon untuk memanggil sumber daya lambda Anda. Kebijakan berbasis sumber daya akan dibuat secara otomatis untuk fungsi lambda yang Anda pilih. Untuk informasi selengkapnya tentang izin, lihat Kebijakan Berbasis Sumber Daya untuk Obrolan Amazon. IVS

Menyiapkan Log Pesan (Opsional)

1. Di bagian Pembuatan log pesan, pilih Buat log pesan obrolan secara otomatis. Bagian Pembuatan log diperluas untuk menampilkan opsi tambahan. Anda dapat menambahkan konfigurasi pembuatan log yang sudah ada ke ruang ini atau membuat konfigurasi pembuatan log baru dengan memilih Buat konfigurasi pembuatan log.
2. Jika Anda memilih konfigurasi pembuatan log yang sudah ada, menu tarik-turun akan muncul dan menampilkan semua konfigurasi pembuatan log yang sudah Anda buat. Pilih salah satu dari daftar tersebut dan log pesan obrolan Anda akan secara otomatis dibuat di tujuan ini.

3. Jika Anda memilih Buat konfigurasi pembuatan log, jendela modal akan muncul agar Anda dapat membuat dan menyesuaikan konfigurasi pembuatan log baru.
   a. Atau, tentukan Nama konfigurasi pembuatan log. Nama konfigurasi log, seperti nama kamar, tidak unik, tetapi mereka menyediakan cara bagi Anda untuk membedakan konfigurasi logging selain konfigurasi logging. ARN
   b. Di bagian Tujuan, pilih grup CloudWatch log, aliran pengiriman firehose Kinesis, atau bucket Amazon S3 untuk memilih tujuan log Anda.
   c. Bergantung pada Tujuan Anda, pilih opsi untuk membuat grup log baru atau gunakan grup CloudWatch log yang sudah ada, aliran pengiriman Firehose Kinesis, atau bucket Amazon S3.
   d. Setelah meninjau, pilih Buat untuk membuat konfigurasi logging baru dengan unikARN. Hal ini secara otomatis akan melampirkan konfigurasi pembuatan log baru ke ruang obrolan.

Pembuatan Ruang Akhir

1. Setelah meninjau, pilih Buat ruang obrolan untuk membuat ruang obrolan baru dengan unikARN.

CLIPetunjuk untuk Membuat Ruang IVS Obrolan

Dokumen ini membawa Anda melalui langkah-langkah yang terlibat dalam membuat ruang IVS obrolan Amazon menggunakan AWSCLI.

Membuat Ruang Obrolan

Membuat ruang obrolan dengan opsi lanjutan dan mengharuskan Anda mengunduh dan mengonfigurasi terlebih dahulu CLI di mesin Anda. AWS CLI Untuk detailnya, lihat Panduan Pengguna Antarmuka Baris AWS Perintah.

1. Jalankan perintah `create-room` obrolan dan berikan nama opsional:

```
aws ivschat create-room --name test-room
```

2. Langkah ini akan mengembalikan ruang obrolan baru:

```
{
   "arn": "arn:aws:ivschat:us-west-2:123456789012:room/g1H2I3j4k5L6",
   "id": "string",
   "createTime": "2021-06-07T14:26:05-07:00",
   "maximumMessageLength": 200,
```

```
    "maximumMessageRatePerSecond": 10,
    "name": "test-room",
    "tags": {},
    "updateTime": "2021-06-07T14:26:05-07:00"
}
```

3. Perhatikan bidang `arn`. Anda akan memerlukan ini untuk membuat token klien dan terhubung ke ruang obrolan.

Menyiapkan Konfigurasi Pembuatan Log (Opsional)

Seperti halnya membuat ruang obrolan, menyiapkan konfigurasi logging dengan opsi lanjutan dan mengharuskan Anda mengunduh dan mengonfigurasi terlebih dahulu CLI di mesin Anda. AWS CLI Untuk detailnya, lihat Panduan Pengguna Antarmuka Baris AWS Perintah.

1. Jalankan perintah `create-logging-configuration` obrolan dan berikan nama opsional serta konfigurasi tujuan yang mengarah ke bucket Amazon S3 berdasarkan nama. Bucket Amazon S3 ini harus sudah ada sebelum membuat konfigurasi pembuatan log. (Untuk detail tentang pembuatan bucket Amazon S3, lihat Dokumentasi Amazon S3.)

```
aws ivschat create-logging-configuration \
   --destination-configuration s3={bucketName=demo-logging-bucket} \
   --name "test-logging-config"
```

2. Langkah ini akan mengembalikan konfigurasi pembuatan log baru:

```
{
   "Arn": "arn:aws:ivschat:us-west-2:123456789012:logging-configuration/
ABcdef34ghIJ",
   "createTime": "2022-09-14T17:48:00.653000+00:00",
   "destinationConfiguration": {
      "s3": {"bucketName": "demo-logging-bucket"}
   },
   "id": "ABcdef34ghIJ",
   "name": "test-logging-config",
   "state": "ACTIVE",
   "tags": {},
   "updateTime": "2022-09-14T17:48:01.104000+00:00"
}
```

3. Perhatikan bidang `arn`. Anda memerlukan bidang ini untuk melampirkan konfigurasi pembuatan log ke ruang obrolan.

   a. Jika Anda membuat ruang obrolan baru, jalankan perintah `create-room` dan berikan `arn` konfigurasi pembuatan log:

   ```
   aws ivschat create-room --name test-room \
   --logging-configuration-identifiers \
   "arn:aws:ivschat:us-west-2:123456789012:logging-configuration/ABcdef34ghIJ"
   ```

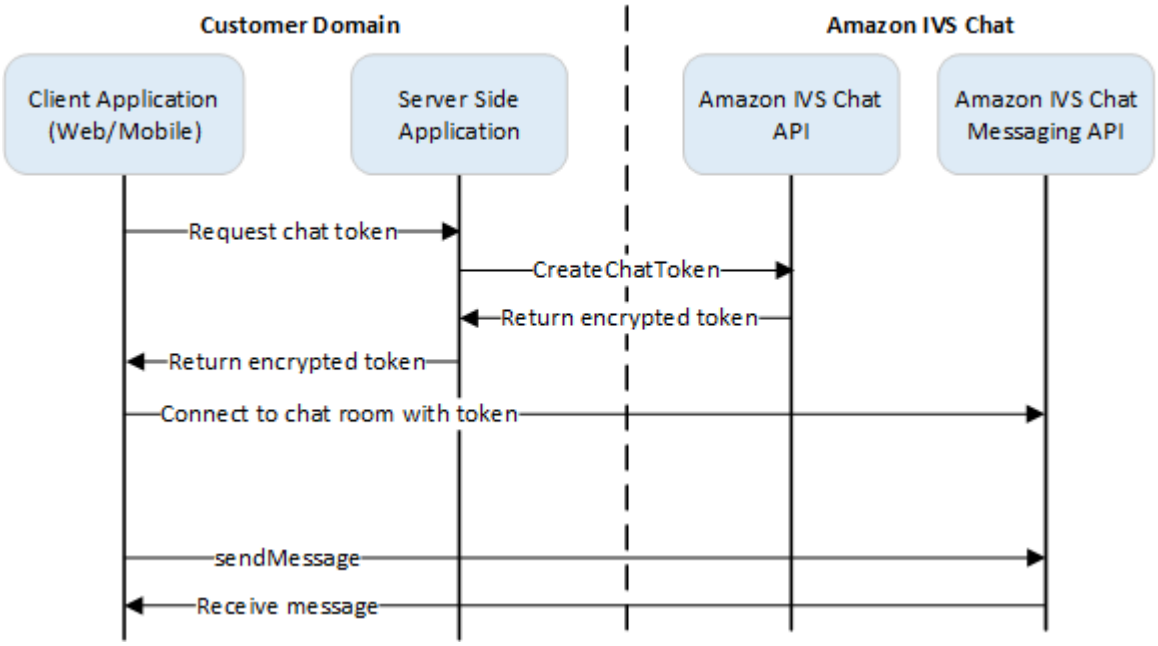
   b. Jika Anda memperbarui ruang obrolan yang sudah ada, jalankan perintah `update-room` dan berikan konfigurasi pembuatan log `arn`:

   ```
   aws ivschat update-room --identifier \
   "arn:aws:ivschat:us-west-2:12345689012:room/g1H2I3j4k5L6" \
   --logging-configuration-identifiers \
   "arn:aws:ivschat:us-west-2:123456789012:logging-configuration/ABcdef34ghIJ"
   ```

Langkah 3: Buat Token Obrolan

Agar peserta obrolan dapat terhubung ke ruang dan mulai mengirim serta menerima pesan, token obrolan harus dibuat. Token obrolan digunakan untuk mengautentikasi dan mengotorisasi klien obrolan.

Diagram ini menggambarkan alur kerja untuk membuat token IVS obrolan:



Seperti yang ditunjukkan di atas, aplikasi klien meminta aplikasi sisi server Anda untuk token, dan aplikasi sisi server memanggil CreateChatToken menggunakan permintaan yang ditandatangani atau SigV4. AWS SDK Karena AWS kredensional digunakan untuk memanggilAPI, token harus dibuat dalam aplikasi sisi server yang aman, bukan aplikasi sisi klien.

Aplikasi server backend yang menunjukkan pembuatan token tersedia di Amazon IVS Chat Demo Backend.

Durasi sesi mengacu pada lama waktu sesi yang ditetapkan dapat tetap aktif sebelum ditutup secara otomatis. Artinya, durasi sesi adalah lama waktu klien dapat tetap terhubung ke ruang obrolan sebelum token baru harus dihasilkan dan koneksi baru harus dibuat. Atau, Anda dapat menentukan durasi sesi selama pembuatan token.

Setiap token hanya dapat digunakan sekali untuk membuat koneksi bagi pengguna akhir. Jika koneksi ditutup, token baru harus dibuat sebelum koneksi dapat dibuat kembali. Token itu sendiri valid hingga stempel waktu kedaluwarsa token yang disertakan dalam respons.

Jika pengguna akhir ingin terhubung ke ruang obrolan, klien harus meminta token ke aplikasi server. Aplikasi server membuat token dan memberikannya kembali ke klien. Token harus dibuat untuk pengguna akhir sesuai permintaan.

Untuk membuat token autentikasi obrolan, ikuti instruksi di bawah ini. Saat Anda membuat token obrolan, gunakan bidang permintaan untuk meneruskan data tentang pengguna akhir obrolan dan

kemampuan pesan pengguna akhir; untuk detailnya, lihat CreateChatTokendi APIReferensi IVS Obrolan.

AWSSDKInstruksi

Membuat token obrolan dengan AWS SDK mengharuskan Anda mengunduh dan mengonfigurasi aplikasi Anda terlebih dahulu. SDK Di bawah ini adalah petunjuk AWS SDK penggunaan JavaScript.

Penting: Kode ini harus dijalankan di sisi server dan output-nya diberikan kepada klien.

Prasyarat: Untuk menggunakan contoh kode di bawah ini, Anda perlu memuat ke AWS JavaScript SDK dalam aplikasi Anda. Untuk detailnya, lihat Memulai dengan AWS SDK for JavaScript.

```
async function createChatToken(params) {
  const ivs = new AWS.Ivschat();
  const result = await ivs.createChatToken(params).promise();
  console.log("New token created", result.token);
}
/*
Create a token with provided inputs. Values for user ID and display name are
from your application and refer to the user connected to this chat session.
*/
const params = {
  "attributes": {
    "displayName": "DemoUser",
  }",
  "capabilities": ["SEND_MESSAGE"],
  "roomIdentifier": "arn:aws:ivschat:us-west-2:123456789012:room/g1H2I3j4k5L6",
  "userId": 11231234
};
createChatToken(params);
```

CLIInstruksi

Membuat token obrolan dengan AWS CLI ini adalah opsi lanjutan dan mengharuskan Anda mengunduh dan mengonfigurasi terlebih dahulu CLI di mesin Anda. Untuk detailnya, lihat Panduan Pengguna Antarmuka Baris AWS Perintah. Catatan: menghasilkan token dengan AWS CLI baik untuk tujuan pengujian, tetapi untuk penggunaan produksi, kami menyarankan Anda membuat token di sisi server dengan AWS SDK (lihat instruksi di atas).

1. Jalankan perintah `create-chat-token` beserta dengan pengidentifikasi ruang dan ID pengguna untuk klien. Sertakan salah satu kemampuan berikut: `"SEND_MESSAGE"`, `"DELETE_MESSAGE"`

,"DISCONNECT_USER". (Atau, sertakan durasi sesi (dalam menit) dan/atau atribut kustom (metadata) tentang sesi obrolan ini. Bidang ini tidak ditampilkan di bawah ini.)

```
aws ivschat create-chat-token --room-identifier "arn:aws:ivschat:us-
west-2:123456789012:room/g1H2I3j4k5L6" --user-id "11231234" --capabilities
 "SEND_MESSAGE"
```

2. Langkah ini akan mengembalikan token klien:

```
{
  "token":
 "abcde12345FGHIJ67890_klmno1234PQRS567890uvwxyz1234.abcd12345EFGHI67890_jklmno123PQRS567890u
  "sessionExpirationTime": "2022-03-16T04:44:09+00:00",
  "tokenExpirationTime": "2022-03-16T03:45:09+00:00"
}
```

3. Simpan token ini. Anda akan memerlukannya untuk terhubung ke ruang obrolan dan mengirim atau menerima pesan. Anda perlu menghasilkan token obrolan lain sebelum sesi Anda berakhir (seperti yang ditunjukkan oleh `sessionExpirationTime`).

Langkah 4: Kirim dan Terima Pesan Pertama Anda

Gunakan token obrolan Anda untuk terhubung ke ruang obrolan dan mengirim pesan pertama. Contoh JavaScript kode disediakan di bawah ini. IVSklien SDKs juga tersedia: lihat ObrolanSDK: Panduan Android, ObrolanSDK: Panduan iOS, dan ObrolanSDK: JavaScript Panduan.

Layanan wilayah: Contoh kode di bawah ini mengacu pada “wilayah pilihan yang didukung.” IVSObrolan Amazon menawarkan titik akhir regional yang dapat Anda gunakan untuk membuat permintaan. Untuk Amazon IVS Chat MessagingAPI, sintaks umum dari endpoint regional adalah:

wss://edge.ivschat.<region-code>.amazonaws.com

Sebagai contoh, wss://edge.ivschat.us-west-2.amazonaws.com adalah titik akhir di wilayah AS Barat (Oregon). Untuk daftar wilayah yang didukung, lihat informasi IVS Obrolan Amazon di IVShalaman Amazon di Referensi AWS Umum.

```
/*
1. To connect to a chat room, you need to create a Secure-WebSocket connection
using the client token you created in the previous steps. Use one of the provided
```

```
endpoints in the Chat Messaging API, depending on your AWS region.
*/
const chatClientToken = "GENERATED_CHAT_CLIENT_TOKEN_HERE";
const socket = "wss://edge.ivschat.us-west-2.amazonaws.com"; // Replace “us-west-2”
 with supported region of choice.
const connection = new WebSocket(socket, chatClientToken);

/*
2. You can send your first message by listening to user input
in the UI and sending messages to the WebSocket connection.
*/
const payload = {
  "Action": "SEND_MESSAGE",
  "RequestId": "OPTIONAL_ID_YOU_CAN_SPECIFY_TO_TRACK_THE_REQUEST",
  "Content": "text message",
  "Attributes": {
    "CustomMetadata": "test metadata"
  }
}
connection.send(JSON.stringify(payload));

/*
3. To listen to incoming chat messages from this WebSocket connection
and display them in your UI, you must add some event listeners.
*/
connection.onmessage = (event) => {
  const data = JSON.parse(event.data);
  displayMessages({
    display_name: data.Sender.Attributes.DisplayName,
    message: data.Content,
    timestamp: data.SendTime
  });
}

function displayMessages(message) {
  // Modify this function to display messages in your chat UI however you like.
  console.log(message);
}

/*
4. Delete a chat message by sending the DELETE_MESSAGE action to the WebSocket
connection. The connected user must have the "DELETE_MESSAGE" permission to
perform this action.
*/
```

```
function deleteMessage(messageId) {
  const deletePayload = {
    "Action": "DELETE_MESSAGE",
    "Reason": "Deleted by moderator",
    "Id": "${messageId}"
  }
  connection.send(deletePayload);
}
```

Selamat, Anda sudah siap! Anda sekarang memiliki aplikasi obrolan sederhana yang dapat mengirim atau menerima pesan.

Langkah 5: Periksa Batas Service Quotas Anda (Opsional)

Ruang obrolan Anda akan diskalakan bersama dengan streaming IVS langsung Amazon Anda, untuk memungkinkan semua pemirsa Anda terlibat dalam percakapan obrolan. Namun, semua IVS akun Amazon memiliki batasan jumlah peserta obrolan bersamaan dan tingkat pengiriman pesan.

Pastikan batas Anda memadai dan minta penambahan jika diperlukan, terutama jika Anda merencanakan acara streaming besar. Untuk detailnya, lihat Service Quotas (Streaming Latensi Rendah), Service Quotas (Streaming Waktu Nyata), dan Service Quotas (Obrolan).

IVSPencatatan Obrolan

Fitur Pencatatan Obrolan memungkinkan Anda merekam semua pesan di ruangan ke salah satu dari tiga lokasi standar: bucket Amazon S3, Amazon CloudWatch Logs, atau Amazon Kinesis Data Firehose. Kemudian, log tersebut dapat digunakan untuk analisis atau membuat tayangan ulang obrolan yang tertaut ke sesi video langsung.

Mengaktifkan Pembuatan Log Obrolan untuk Ruang

Pembuatan Log Obrolan adalah opsi lanjutan yang dapat diaktifkan dengan mengaitkan konfigurasi pembuatan log dengan suatu ruang. Konfigurasi logging adalah sumber daya yang memungkinkan Anda menentukan jenis lokasi (bucket Amazon S3, Amazon CloudWatch Log, atau Amazon Kinesis Data Firehose) tempat pesan ruangan dicatat. Untuk detail tentang membuat dan mengelola konfigurasi logging, lihat Memulai IVS Obrolan Amazon dan APIReferensi IVS Obrolan Amazon.

Anda dapat mengaitkan hingga tiga konfigurasi logging dengan setiap kamar, baik saat membuat kamar baru (CreateRoom) atau memperbarui kamar yang ada (UpdateRoom). Anda dapat mengaitkan banyak ruang dengan konfigurasi pembuatan log yang sama.

Jika setidaknya satu konfigurasi logging aktif dikaitkan dengan sebuah ruangan, setiap permintaan pesan yang dikirim ke ruangan tersebut melalui Pesan IVS Obrolan Amazon API secara otomatis direkam ke lokasi yang ditentukan. Berikut adalah penundaan propagasi rata-rata (dari saat permintaan perpesanan dikirim sampai saat permintaan tersedia di lokasi yang Anda tentukan):

- Bucket Amazon S3: 5 menit
- CloudWatch Log Amazon atau Amazon Kinesis Data Firehose: 10 detik

Isi Pesan

format

```
{
   "event_timestamp": "string",
   "type": "string",
   "version": "string",
   "payload": { "string": "string" }
```

```
}
```

Bidang

Bidang	Deskripsi
`event_timestamp`	UTCstempel waktu kapan pesan diterima oleh Amazon IVS Chat.
`payload`	JSONMuatan Pesan (Berlangganan) atau Acara (Berlangganan) yang akan diterima klien dari layanan IVS Obrolan Amazon.
`type`	Tipe pesan obrolan. • Nilai Valid: `MESSAGE` | `EVENT`
`version`	Versi format konten pesan.

Bucket Amazon S3

format

Log pesan diatur dan disimpan dengan prefiks S3 dan format file berikut:

```
AWSLogs/<account_id>/IVSChatLogs/<version>/<region>/room_<resource_id>/<year>/<month>/
<day>/<hours>/
<account_id>_IVSChatLogs_<version>_<region>_room_<resource_id>_<year><month><day><hours><minute
```

Bidang

Bidang	Deskripsi
`<account_id>`	AWSID akun dari mana ruangan dibuat.
`<hash>`	Nilai hash yang dihasilkan oleh sistem untuk memastikan keunikan.

Bidang	Deskripsi
`<region>`	Wilayah AWS layanan tempat ruangan itu dibuat.
`<resource_id>`	Bagian ID sumber daya ruanganARN.
`<version>`	Versi format konten pesan.
`<year> / <month> / <day> / <hours> / <minute>`	UTCstempel waktu kapan pesan diterima oleh Amazon IVS Chat.

Contoh

```
AWSLogs/123456789012/IVSChatLogs/1.0/us-west-2/
room_abc123DEF456/2022/10/14/17/123456789012_IVSChatLogs_1.0_us-
west-2_room_abc123DEF456_20221014T1740Z_1766dcbc.log.gz
```

CloudWatch Log Amazon

Format

Log pesan diatur dan disimpan dengan format nama log-stream berikut:

```
aws/IVSChatLogs/<version>/room_<resource_id>
```

Bidang

Bidang	Deskripsi
`<resource_id>`	ID sumber daya bagian ruanganARN.
`<version>`	Versi format konten pesan.

Contoh

```
aws/IVSChatLogs/1.0/room_abc123DEF456
```

Amazon Kinesis Data Firehose

Log pesan dikirim ke aliran pengiriman sebagai data streaming real-time ke tujuan seperti Amazon Redshift, Amazon OpenSearch Service, Splunk, dan titik akhir atau HTTP titik akhir kustom apa pun yang dimiliki oleh penyedia layanan HTTP pihak ketiga yang didukung. Untuk informasi selengkapnya, lihat Apa itu Amazon Kinesis Data Firehose.

Batasan

- Anda harus memiliki lokasi pembuatan log tempat pesan akan disimpan.
- Ruang, konfigurasi logging, dan lokasi logging harus berada di AWS wilayah yang sama.
- Hanya konfigurasi pembuatan log aktif yang tersedia untuk Pembuatan Log Obrolan.
- Anda hanya dapat menghapus konfigurasi pembuatan log yang tidak lagi terkait dengan ruang mana pun.

Mencatat pesan ke lokasi yang Anda miliki memerlukan otorisasi dengan AWS kredensi Anda. Untuk memberikan akses yang diperlukan kepada IVS Chat, kebijakan sumber daya (untuk bucket atau CloudWatch Log Amazon S3) atau Peran AWS IAM Tertaut Layanan (SLR) (untuk Amazon Kinesis Data Firehose) dibuat secara otomatis saat konfigurasi logging dibuat. Berhati-hatilah dengan modifikasi apa pun pada peran atau kebijakan, karena hal tersebut dapat memengaruhi izin untuk pembuatan log obrolan.

Memantau Kesalahan dengan Amazon CloudWatch

Anda dapat memantau kesalahan yang terjadi dalam pencatatan obrolan dengan Amazon CloudWatch, dan Anda dapat membuat alarm atau dasbor untuk menunjukkan atau menanggapi perubahan kesalahan tertentu.

Ada beberapa tipe kesalahan. Untuk informasi selengkapnya, lihat Memantau IVS Obrolan Amazon.

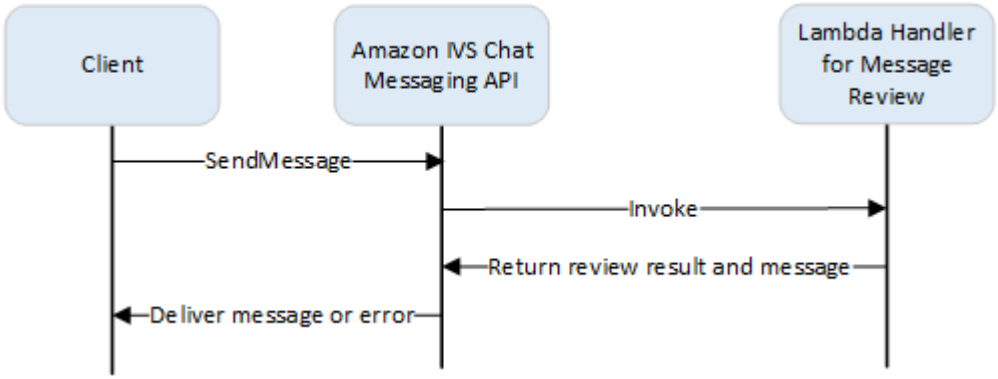
IVSPenangan Ulasan Pesan Obrolan

Handler tinjauan pesan memungkinkan Anda untuk meninjau dan/atau memodifikasi pesan sebelum dikirim ke suatu ruang. Ketika peninjau pesan dikaitkan dengan sebuah ruangan, penangan tersebut dipanggil untuk setiap SendMessage permintaan ke ruangan itu. Handler memberlakukan logika bisnis aplikasi Anda dan menentukan apakah akan mengizinkan, menolak, atau memodifikasi pesan. IVSObrolan Amazon mendukung fungsi AWS Lambda sebagai penangan.

Membuat fungsi Lambda

Sebelum menyiapkan peninjau pesan untuk sebuah ruangan, Anda harus membuat fungsi lambda dengan kebijakan berbasis sumber daya. IAM Fungsi lambda harus berada di AWS akun dan AWS wilayah yang sama dengan ruangan tempat Anda akan menggunakan fungsi tersebut. Kebijakan berbasis sumber daya memberikan izin IVS Obrolan Amazon untuk menjalankan fungsi lambda Anda. Untuk petunjuk, lihat Kebijakan Berbasis Sumber Daya untuk Obrolan Amazon. IVS

Alur kerja



Sintaks Permintaan

Saat klien mengirim pesan, Amazon IVS Chat akan memanggil fungsi lambda dengan payload: JSON

```
{
   "Content": "string",
   "MessageId": "string",
   "RoomArn": "string",
   "Attributes": {"string": "string"},
   "Sender": {
```

```
      "Attributes": { "string": "string" },
      "UserId": "string",
      "Ip": "string"
   }
}
```

Isi Permintaan

Bidang	Deskripsi
`Attributes`	Atribut yang terkait dengan pesan.
`Content`	Konten asli dari pesan.
`MessageId`	ID pesan. Dihasilkan oleh IVS Obrolan.
`RoomArn`	Ruang tempat pesan dikirim. ARN
`Sender`	Informasi tentang pengirim. Objek ini memiliki beberapa bidang: • `Attributes` — Metadata tentang pengirim yang dibuat selama autentikasi. Ini dapat digunakan untuk memberikan klien informasi lebih lanjut tentang pengirim; misalnya, avatar, lencanaURL, font, dan warna. • `UserId` — Pengidentifikasi yang ditentukan aplikasi dari penampil (pengguna akhir) yang mengirim pesan ini. Ini dapat digunakan oleh aplikasi klien untuk merujuk ke pengguna baik di domain perpesanan API atau aplikasi. • `Ip` — Alamat IP klien yang mengirim pesan.

Sintaks Respons

Fungsi handler lambda harus mengembalikan JSON respons dengan sintaks berikut. Respons yang tidak sesuai dengan sintaks di bawah ini atau memenuhi batasan bidang akan menjadi tidak valid. Dalam hal ini, pesan diizinkan atau ditolak tergantung pada `FallbackResult` nilai yang Anda tentukan dalam peninjau pesan; lihat MessageReviewHandlerdi APIReferensi IVS Obrolan Amazon.

```
{
   "Content": "string",
```

```
   "ReviewResult": "string",
   "Attributes": {"string": "string"},
}
```

Bidang Respons

Bidang	Deskripsi
`Attributes`	Atribut yang terkait dengan pesan yang dikembalikan dari fungsi lambda. Jika `ReviewResult` adalah `DENY`, `Reason` dapat disediakan di `Attributes` ; misalnya: `"Attributes": {"Reason": "denied for moderation` Dalam hal ini, klien pengirim menerima kesalahan WebSocket 406 dengan alasan dalam pesan kesalahan. (Lihat WebSocket Kesalahan dalam APIReferensi Pesan IVS Obrolan Amazon.) • Batasan Ukuran: Maksimum 1 KB • Wajib: Tidak
`Content`	Konten pesan dikembalikan dari fungsi Lambda. Pesan tersebut dapat diedit atau tetap seperti aslinya tergantung pada logika bisnis. • Batasan Panjang: Panjang minimum 1. Panjang maksimum `MaximumMessageLength` yang Anda tentukan saat Anda membuat/memperbarui ruang. Lihat APIReferensi IVS Obrolan Amazon untuk informasi selengkapnya. Ketentuan tersebut hanya berlaku jika `ReviewResult` adalah `ALLOW`. • Wajib: Ya
`ReviewResult`	Hasil pemrosesan tinjauan tentang cara menangani pesan. Jika diizinkan, pesan dikirimkan ke semua pengguna yang terhubung ke ruang. Jika ditolak, pesan tidak dikirimkan ke pengguna mana pun. • Nilai Valid: `ALLOW` | `DENY` • Wajib: Ya

Kode Sampel

Di bawah ini adalah contoh handler lambda di Go. Handler memodifikasi konten pesan, menjaga atribut pesan agar tidak berubah, dan mengizinkan pesan.

```
package main

import (
   "context"
   "github.com/aws/aws-lambda-go/lambda"
)

type Request struct {
   MessageId string
   Content string
   Attributes map[string]string
   RoomArn string
   Sender Sender
}

type Response struct {
   ReviewResult string
   Content string
   Attributes map[string]string
}

type Sender struct {
   UserId string
   Ip string
   Attributes map[string]string
}

func main() {
   lambda.Start(HandleRequest)
}

func HandleRequest(ctx context.Context, request Request) (Response, error) {
   content := request.Content + "modified by the lambda handler"
   return Response{
       ReviewResult: "ALLOW",
       Content: content,
   }, nil
}
```

Mengaitkan dan Memutuskan Kaitan Handler dengan Ruang

Setelah handler lambda disiapkan dan diimplementasikan, gunakan Obrolan Amazon IVS: API

- Untuk mengaitkan handler dengan ruangan, hubungi CreateRoom atau UpdateRoom dan tentukan handler.
- Untuk memisahkan handler dari sebuah ruangan, panggil UpdateRoom dengan nilai kosong untuk. `MessageReviewHandler.Uri`

Memantau Kesalahan dengan Amazon CloudWatch

Anda dapat memantau kesalahan yang terjadi dalam peninjauan pesan dengan Amazon CloudWatch, dan Anda dapat membuat alarm atau dasbor untuk menunjukkan atau menanggapi perubahan kesalahan tertentu. Jika terjadi kesalahan, pesan diizinkan atau ditolak tergantung pada `FallbackResult` nilai yang Anda tentukan saat Anda mengaitkan handler dengan ruangan; lihat MessageReviewHandlerdi APIReferensi IVS Obrolan Amazon.

Ada beberapa tipe kesalahan:

- `InvocationErrors`terjadi ketika IVS Obrolan Amazon tidak dapat memanggil penangan.
- `ResponseValidationErrors` terjadi saat handler mengembalikan respons yang tidak valid.
- AWSLambda `Errors` terjadi ketika penangan lambda mengembalikan kesalahan fungsi ketika telah dipanggil.

Untuk informasi selengkapnya tentang kesalahan pemanggilan dan kesalahan validasi respons (yang dipancarkan oleh Obrolan AmazonIVS), lihat Memantau Obrolan Amazon. IVS Untuk informasi selengkapnya tentang kesalahan AWS Lambda, lihat Bekerja dengan Metrik Lambda.

Memantau IVS Obrolan Amazon

Anda dapat memantau sumber daya Obrolan Amazon Interactive Video Service (IVS) menggunakan Amazon CloudWatch. CloudWatch mengumpulkan dan memproses data mentah dari Amazon IVS Chat menjadi metrik hampir real-time yang dapat dibaca. Statistik ini disimpan selama 15 bulan, sehingga Anda bisa mendapatkan perspektif historis tentang performa aplikasi atau layanan web Anda. Anda dapat mengatur alarm untuk ambang batas tertentu dan mengirimkan notifikasi atau mengambil tindakan saat ambang batas tersebut terpenuhi. Untuk detailnya, lihat Panduan CloudWatch Pengguna.

CloudWatch Metrik Akses

Amazon CloudWatch mengumpulkan dan memproses data mentah dari Amazon IVS Chat menjadi metrik yang dapat dibaca. near-real-time Statistik ini disimpan selama 15 bulan, sehingga Anda bisa mendapatkan perspektif historis tentang performa aplikasi atau layanan web Anda. Anda dapat mengatur alarm untuk ambang batas tertentu dan mengirimkan notifikasi atau mengambil tindakan saat ambang batas tersebut terpenuhi. Untuk detailnya, lihat Panduan CloudWatch Pengguna.

Perhatikan bahwa CloudWatch metrik digulung dari waktu ke waktu. Resolusi menurun secara efektif seiring bertambahnya usia metrik. Berikut jadwalnya:

- Metrik 60 detik tersedia selama 15 hari.
- Metrik 5 menit tersedia selama 63 hari.
- Metrik 1 jam tersedia selama 455 hari (15 bulan).

Untuk informasi terkini tentang retensi data, cari “periode retensi” di Amazon CloudWatch FAQs.

CloudWatch Petunjuk Konsol

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Di navigasi samping, perluas menu tarik-turun Metrik, lalu pilih Semua metrik.
3. Pada tab Jelajahi, menggunakan menu tarik-turun tidak berlabel di sebelah kiri, pilih wilayah “asal” Anda, tempat saluran dibuat. Untuk informasi selengkapnya tentang wilayah, lihat Solusi Global, Kontrol Regional. Untuk daftar wilayah yang didukung, lihat IVShalaman Amazon di Referensi AWS Umum.

4. Di bagian bawah tab Browse, pilih IVSChatnamespace.
5. Lakukan salah satu hal berikut ini:
   a. Di bilah pencarian, masukkan ID sumber daya Anda (bagian dariARN,`arn:::ivschat:room/<resource id>`).

      Kemudian pilih IVSChat.
   b. Jika IVSChatmuncul sebagai layanan yang dapat dipilih di bawah AWSNamespaces, pilih itu. Ini akan terdaftar jika Anda menggunakan Amazon IVSChat dan mengirim metrik ke Amazon CloudWatch. (Jika IVSChattidak terdaftar, Anda tidak memiliki IVSChat metrik Amazon.)

      Kemudian pilih pengelompokan dimensi sesuai keinginan; dimensi yang tersedia tercantum dalam CloudWatch Metrik di bawah ini.
6. Pilih metrik yang akan ditambahkan ke grafik. Metrik yang tersedia tercantum dalam CloudWatch Metrik di bawah ini.

Anda juga dapat mengakses CloudWatch bagan sesi obrolan Anda dari halaman detail sesi obrolan, dengan memilih CloudWatch tombol Lihat di.

CLIInstruksi

Anda juga dapat mengakses metrik menggunakan. AWS CLI Ini mengharuskan Anda mengunduh dan mengkonfigurasi terlebih dahulu CLI di mesin Anda. Untuk detailnya, lihat Panduan Pengguna Antarmuka Baris AWS Perintah.

Kemudian, untuk mengakses metrik obrolan IVS latensi rendah Amazon menggunakan: AWS CLI

- Di prompt perintah, jalankan:

```
aws cloudwatch list-metrics --namespace AWS/IVSChat
```

Untuk informasi selengkapnya, lihat Menggunakan CloudWatch Metrik Amazon di Panduan CloudWatch Pengguna Amazon.

CloudWatch Metrik: IVS Obrolan

Amazon IVS Chat menyediakan metrik berikut di IVSChat namespace AWS/.

Metrik	Dimensi	Deskripsi
`ConcurrentChatConnections`	Tidak ada	Jumlah total koneksi bersamaan di ruang obrolan (dilaporkan maks per menit). Jumlah tersebut berguna untuk memahami waktu pelanggan mendekati batas mereka untuk koneksi obrolan bersamaan di suatu wilayah. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`Deliveries`	Tindakan	Jumlah pengiriman permintaan perpesanan yang dibuat dari tipe tindakan tertentu untuk koneksi obrolan di seluruh ruang Anda di suatu wilayah. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`InvocationErrors`	`Uri`	Jumlah kesalahan invokasi handler tinjauan pesan tertentu di seluruh ruang Anda di suatu wilayah. Terjadi kesalahan invokasi saat handler tinjauan pesan tidak dapat diinvokasi. Kesalahan pemanggilan terjadi ketika IVS Obrolan Amazon tidak dapat memanggil penangan. Hal tersebut dapat terjadi jika handler yang terkait dengan ruang tidak ada lagi atau kehabisan waktu, atau jika kebijakan sumber dayanya tidak mengizinkan layanan untuk menginvokasinya. Unit: Jumlah

Metrik	Dimensi	Deskripsi
		Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`LogDestinationAccessDeniedError`	`LoggingConfiguration`	Jumlah kesalahan akses yang ditolak dari tujuan log di semua ruang Anda di suatu wilayah. Kesalahan ini terjadi ketika Amazon IVS Chat tidak dapat mengakses sumber daya tujuan yang Anda tentukan dalam konfigurasi logging. Hal tersebut dapat terjadi jika kebijakan sumber daya tujuan tidak mengizinkan layanan untuk menaruh catatan. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`LogDestinationErrors`	`LoggingConfiguration`	Jumlah semua kesalahan dari tujuan log di semua ruang Anda di suatu wilayah. Ini adalah metrik agregat yang mencakup semua jenis kesalahan yang terjadi saat Amazon IVS Chat gagal mengirimkan log ke sumber daya tujuan yang Anda tentukan dalam konfigurasi logging. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum

Metrik	Dimensi	Deskripsi
`LogDestinationResourceNotFoundErrors`	`LoggingConfiguration`	Jumlah resource-not-found kesalahan tujuan log di semua kamar Anda di suatu wilayah. Kesalahan ini terjadi ketika Amazon IVS Chat tidak dapat mengirimkan log ke sumber daya tujuan yang Anda tentukan dalam konfigurasi logging karena sumber daya tidak ada. Hal ini dapat terjadi jika sumber daya tujuan yang terkait dengan konfigurasi pembuatan log tidak ada lagi. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`MessagingDeliveries`	Tidak ada	Jumlah pengiriman permintaan perpesanan ke koneksi obrolan di seluruh ruang Anda di suatu wilayah. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`MessagingRequests`	Tidak ada	Jumlah permintaan perpesanan yang dibuat di seluruh ruang Anda di suatu wilayah. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum

Metrik	Dimensi	Deskripsi
`Requests`	Tindakan	Jumlah permintaan yang dibuat dari tipe tindakan tertentu di seluruh ruang Anda di suatu wilayah. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum
`ResponseValidationErrors`	`Uri`	Jumlah kesalahan validasi respons dari handler tinjauan pesan tertentu di semua ruang Anda di suatu wilayah. Kesalahan validasi respons terjadi ketika respons dari handler tinjauan pesan tidak valid. Hal tersebut mungkin berarti bahwa respons tidak dapat diuraikan atau gagal dalam pemeriksaan validasi; misalnya, hasil tinjauan tidak valid atau nilai respons terlalu panjang. Unit: Jumlah Statistik yang valid: Jumlah, Rata-rata, Maksimum, Minimum

IVSPesan Klien Obrolan SDK

Amazon Interactive Video Services (IVS) Chat Client Messaging SDK adalah untuk pengembang yang sedang membangun aplikasi dengan AmazonIVS. Ini SDK dirancang untuk memanfaatkan IVS arsitektur Amazon dan akan melihat pembaruan, di samping IVS Obrolan Amazon. Sebagai asliSDK, ini dirancang untuk meminimalkan dampak kinerja pada aplikasi Anda dan pada perangkat yang digunakan pengguna Anda mengakses aplikasi Anda.

Persyaratan Platform

Peramban Desktop

Peramban	Versi yang Didukung
Chrome	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Edge	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Firefox	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Opera	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Safari	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)

Peramban Seluler

Peramban	Versi yang Didukung
Chrome untuk Android	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Firefox untuk Android	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Opera untuk Android	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)

Peramban	Versi yang Didukung
WebView Android	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Internet Samsung	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)
Safari untuk iOS	Dua versi utama (versi saat ini dan versi terbaru sebelumnya)

Platform Native

Platform	Versi yang Didukung
Android	5.0 dan versi yang lebih baru
iOS	13.0 dan versi yang lebih baru

Dukungan

Jika Anda mengalami kesalahan atau masalah lain dengan ruang obrolan Anda, tentukan pengenal ruang unik melalui IVS Obrolan API (lihat ListRooms).

Bagikan pengenal ruang obrolan ini dengan AWS dukungan. Dengan pengidentifikasi tersebut, mereka bisa mendapatkan informasi untuk membantu memecahkan masalah Anda.

Catatan: Lihat Catatan Rilis IVS Obrolan Amazon untuk versi yang tersedia dan masalah yang diperbaiki. Jika perlu, sebelum menghubungi dukungan, perbarui versi Anda SDK dan lihat apakah itu menyelesaikan masalah Anda.

Penentuan Versi

Pesan Klien IVS Obrolan Amazon SDKs menggunakan versi semantik.

Untuk diskusi ini, misalkan:

- Rilis terbaru adalah 4.1.3.

- Rilis terbaru dari versi utama sebelumnya adalah 3.2.4.
- Rilis terbaru versi 1.x adalah 1.5.6.

Fitur baru yang kompatibel dengan versi sebelumnya ditambahkan sebagai rilis minor dari versi terbaru. Dalam hal ini, rangkaian fitur baru berikutnya akan ditambahkan sebagai versi 4.2.0.

Perbaikan bug minor yang kompatibel dengan versi sebelumnya ditambahkan sebagai rilis patch dari versi terbaru. Di sini, set perbaikan bug minor berikutnya akan ditambahkan sebagai versi 4.1.4.

Perbaikan bug besar yang kompatibel dengan versi sebelumnya ditangani secara berbeda; perbaikan ini ditambahkan ke beberapa versi:

- Rilis patch versi terbaru. Di sini, ini adalah versi 4.1.4.
- Rilis patch dari versi kecil sebelumnya. Di sini, ini adalah versi 3.2.5.
- Rilis patch dari rilis versi 1.x terbaru. Di sini, ini adalah versi 1.5.7.

Perbaikan bug utama ditentukan oleh tim IVS produk Amazon. Contoh umumnya adalah pembaruan keamanan yang penting dan perbaikan pilihan lainnya yang diperlukan pelanggan.

Catatan: Dalam contoh di atas, versi yang dirilis meningkat tanpa melewatkan angka apa pun (misalnya, dari 4.1.3 ke 4.1.4). Pada kenyataannya, satu atau beberapa nomor patch mungkin tetap bersifat internal dan tidak dirilis, sehingga versi yang dirilis dapat meningkat dari 4.1.3 menjadi, katakanlah, 4.1.6.

Selain itu, versi 1.x akan didukung hingga akhir tahun 2023 atau saat 3.x dirilis, mana yang lebih dahulu.

IVSObrolan Amazon APIs

Di sisi server (tidak dikelola olehSDKs), ada duaAPIs, masing-masing dengan tanggung jawabnya sendiri:

- Pesawat data — Pesan IVS API Obrolan WebSockets API dirancang untuk digunakan oleh aplikasi front-end (iOS, Android, macOS, dll) yang didorong oleh skema otentikasi berbasis token. Menggunakan token obrolan yang dibuat sebelumnya, Anda terhubung ke ruang obrolan yang sudah ada menggunakan iniAPI.

Pesan SDKs Klien IVS Obrolan Amazon hanya berkaitan dengan bidang data. SDKsAsumsikan bahwa Anda sudah menghasilkan token obrolan melalui backend Anda. Pengambilan token ini diasumsikan dikelola oleh aplikasi front-end Anda, bukan. SDKs

- Pesawat kontrol - Pesawat Kontrol IVS Obrolan API menyediakan antarmuka untuk aplikasi backend Anda sendiri untuk mengelola dan membuat ruang obrolan serta pengguna yang bergabung dengan mereka. Anggap ini sebagai panel admin untuk pengalaman obrolan aplikasi Anda yang dikelola oleh backend Anda sendiri. Ada titik akhir bidang kontrol yang bertanggung jawab untuk membuat token obrolan yang perlu diautentikasi oleh bidang data ke ruang obrolan.

  Penting: Pesan Klien IVS Obrolan SDKs tidak memanggil titik akhir bidang kontrol apa pun. Anda harus menyiapkan backend untuk membuat token obrolan. Aplikasi front-end Anda harus berkomunikasi dengan backend untuk mengambil token obrolan ini.

IVSPesan Klien ObrolanSDK: Panduan Android

Amazon Interactive Video (IVS) Chat Client Messaging Android SDK menyediakan antarmuka yang memungkinkan Anda untuk dengan mudah menggabungkan Pesan IVS Obrolan kami API di platform menggunakan Android.

Paket `com.amazonaws:ivs-chat-messaging` mengimplementasikan antarmuka yang dijelaskan dalam dokumen ini.

Versi terbaru dari IVS Chat Client Messaging AndroidSDK: 1.1.0 (Catatan Rilis)

Dokumentasi referensi: Untuk informasi tentang metode terpenting yang tersedia di Android Pesan Klien IVS Obrolan AmazonSDK, lihat dokumentasi referensi di: https://aws.github.io/ amazon-ivs-chat-messaging -sdk-android/1.1.0/

Contoh kode: Lihat contoh repositori Android di GitHub: -android-demo https://github.com/aws-samples/ amazon-ivs-chat-for

Persyaratan platform: Android 5.0 (APIlevel 21) atau lebih tinggi diperlukan untuk pengembangan.

Memulai dengan IVS Chat Client Messaging Android SDK

Sebelum memulai, Anda harus terbiasa dengan Memulai dengan IVS Obrolan Amazon.

Menambahkan Paket

Tambahkan `com.amazonaws:ivs-chat-messaging` ke dependensi `build.gradle` Anda:

```
dependencies {
   implementation 'com.amazonaws:ivs-chat-messaging'
}
```

Tambahkan Aturan Proguard

Tambahkan entri berikut ke file aturan R8/Proguard Anda (`proguard-rules.pro`):

```
-keep public class com.amazonaws.ivs.chat.messaging.** { *; }
-keep public interface com.amazonaws.ivs.chat.messaging.** { *; }
```

Siapkan Backend Anda

Integrasi ini memerlukan titik akhir di server Anda yang berbicara dengan Amazon IVS API. Gunakan AWSperpustakaan resmi untuk akses ke Amazon IVS API dari server Anda. Pustaka ini dapat diakses dalam beberapa bahasa dari paket publik; misalnya, node.js dan Java.

Selanjutnya, buat endpoint server yang berbicara dengan IVSObrolan Amazon API dan buat token.

Menyiapkan Koneksi Server

Buat metode yang menggunakan `ChatTokenCallback` sebagai param dan mengambil token obrolan dari backend Anda. Teruskan token tersebut ke metode `onSuccess` panggilan balik. Jika terjadi kesalahan, teruskan pengecualian ke metode `onError` dari panggilan balik. Hal ini diperlukan untuk instantiasi entitas `ChatRoom` utama di langkah berikutnya.

Di bawah ini Anda dapat menemukan contoh kode yang mengimplementasikan hal-hal di atas dengan menggunakan panggilan `Retrofit`.

```
// ...

private fun fetchChatToken(callback: ChatTokenCallback) {
    apiService.createChatToken(userId, roomId).enqueue(object : Callback<ChatToken> {
        override fun onResponse(call: Call<ExampleResponse>, response:
 Response<ExampleResponse>) {
            val body = response.body()
            val token = ChatToken(
                body.token,
```

```
                body.sessionExpirationTime,
                body.tokenExpirationTime
            )
            callback.onSuccess(token)
        }

        override fun onFailure(call: Call<ChatToken>, throwable: Throwable) {
            callback.onError(throwable)
        }
    })
}
// ...
```

Menggunakan IVS Chat Client Messaging Android SDK

Dokumen ini membawa Anda melalui langkah-langkah yang terlibat dalam menggunakan pesan klien IVS obrolan Amazon AndroidSDK.

Menginisialisasi Instans Ruang Obrolan

Buat instans dari kelas `ChatRoom`. Ini membutuhkan passing`regionOrUrl`, yang biasanya merupakan AWS wilayah di mana ruang obrolan Anda di-host, dan `tokenProvider` yang merupakan metode pengambilan token yang dibuat pada langkah sebelumnya.

```
val room = ChatRoom(
    regionOrUrl = "us-west-2",
    tokenProvider = ::fetchChatToken
)
```

Selanjutnya, buat objek pendengar yang akan mengimplementasikan handler untuk peristiwa yang terkait obrolan, dan tetapkan objek tersebut ke properti `room.listener`:

```
private val roomListener = object : ChatRoomListener {
    override fun onConnecting(room: ChatRoom) {
      // Called when room is establishing the initial connection or reestablishing
 connection after socket failure/token expiration/etc
    }

    override fun onConnected(room: ChatRoom) {
        // Called when connection has been established
    }
```

```
    override fun onDisconnected(room: ChatRoom, reason: DisconnectReason) {
        // Called when a room has been disconnected
    }

    override fun onMessageReceived(room: ChatRoom, message: ChatMessage) {
        // Called when chat message has been received
    }

    override fun onEventReceived(room: ChatRoom, event: ChatEvent) {
        // Called when chat event has been received
    }

    override fun onDeleteMessage(room: ChatRoom, event: DeleteMessageEvent) {
       // Called when DELETE_MESSAGE event has been received
    }
}

val room = ChatRoom(
    region = "us-west-2",
    tokenProvider = ::fetchChatToken
)

room.listener = roomListener // <- add this line

// ...
```

Langkah terakhir dari inisialisasi dasar adalah menghubungkan ke ruangan tertentu dengan membuat WebSocket koneksi. Untuk melakukan hal tersebut, panggil metode `connect()` dalam instans ruang. Sebaiknya lakukan langkah tersebut dalam metode siklus hidup `onResume()` untuk memastikannya tetap terhubung jika aplikasi Anda dilanjutkan dari latar belakang.

```
room.connect()
```

SDKAkan mencoba membuat koneksi ke ruang obrolan yang dikodekan dalam token obrolan yang diterima dari server Anda. Jika gagal, SDK akan mencoba menghubungkan kembali sebanyak yang ditentukan dalam instans ruang.

Melakukan Tindakan di Ruang Obrolan

Kelas `ChatRoom` memiliki tindakan untuk mengirim dan menghapus pesan serta memutus koneksi pengguna lain. Tindakan ini menerima parameter panggilan balik opsional yang memungkinkan Anda untuk mendapatkan konfirmasi permintaan atau notifikasi penolakan.

Mengirim Pesan

Untuk permintaan ini, Anda harus memiliki kemampuan `SEND_MESSAGE` yang dienkode dalam token obrolan Anda.

Untuk memicu permintaan kirim-pesan:

```
val request = SendMessageRequest("Test Echo")
room.sendMessage(request)
```

Untuk mendapatkan konfirmasi/penolakan permintaan, sediakan panggilan balik sebagai parameter kedua:

```
room.sendMessage(request, object : SendMessageCallback {
   override fun onConfirmed(request: SendMessageRequest, response: ChatMessage) {
      // Message was successfully sent to the chat room.
   }
   override fun onRejected(request: SendMessageRequest, error: ChatError) {
      // Send-message request was rejected. Inspect the `error` parameter for details.
   }
})
```

Menghapus Pesan

Untuk permintaan ini, Anda harus memiliki MESSAGE kemampuan DELETE _ yang dikodekan dalam token obrolan Anda.

Untuk memicu permintaan hapus-pesan:

```
val request = DeleteMessageRequest(messageId, "Some delete reason")
room.deleteMessage(request)
```

Untuk mendapatkan konfirmasi/penolakan permintaan, sediakan panggilan balik sebagai parameter kedua:

```
room.deleteMessage(request, object : DeleteMessageCallback {
   override fun onConfirmed(request: DeleteMessageRequest, response:
 DeleteMessageEvent) {
      // Message was successfully deleted from the chat room.
   }
```

```
   override fun onRejected(request: DeleteMessageRequest, error: ChatError) {
      // Delete-message request was rejected. Inspect the `error` parameter for
 details.
   }
})
```

Memutus Koneksi Pengguna Lain

Untuk permintaan ini, Anda harus memiliki kemampuan `DISCONNECT_USER` yang dienkode dalam token obrolan Anda.

Guna memutus koneksi pengguna lain untuk tujuan moderasi:

```
val request = DisconnectUserRequest(userId, "Reason for disconnecting user")
room.disconnectUser(request)
```

Untuk mendapatkan konfirmasi/penolakan permintaan, sediakan panggilan balik sebagai parameter kedua:

```
room.disconnectUser(request, object : DisconnectUserCallback {
   override fun onConfirmed(request: SendMessageRequest, response: ChatMessage) {
      // User was disconnected from the chat room.
   }
   override fun onRejected(request: SendMessageRequest, error: ChatError) {
      // Disconnect-user request was rejected. Inspect the `error` parameter for
 details.
   }
})
```

Memutus Koneksi dari Ruang Obrolan

Untuk menutup koneksi Anda ke ruang obrolan, panggil metode `disconnect()` pada instans ruang:

```
room.disconnect()
```

Karena WebSocket koneksi akan berhenti bekerja setelah waktu yang singkat ketika aplikasi dalam keadaan latar belakang, kami sarankan Anda secara manual connect/disconnect when transitioning from/to status latar belakang. Untuk melakukannya, pasangkan panggilan `room.connect()` dalam metode siklus hidup `onResume()`, di `Activity` atau `Fragment` Android, dengan panggilan `room.disconnect()` dalam metode siklus hidup `onPause()`.

IVSPesan Klien ObrolanSDK: Tutorial Android Bagian 1: Ruang Obrolan

Ini adalah bagian pertama dari tutorial dua bagian. Anda akan mempelajari hal-hal penting dalam bekerja dengan Amazon IVS Chat Messaging SDK dengan membangun aplikasi Android yang berfungsi penuh menggunakan bahasa pemrograman Kotlin. Kami menyebut aplikasi itu Chatterbox.

Sebelum Anda memulai modul, luangkan beberapa menit untuk membiasakan diri dengan prasyarat, konsep utama di balik token obrolan, dan server backend yang diperlukan untuk membuat ruang obrolan.

Tutorial ini dibuat untuk pengembang Android berpengalaman yang baru mengenal IVS Chat MessagingSDK. Anda harus merasa nyaman dengan bahasa pemrograman Kotlin dan membuat UIs di platform Android.

Bagian pertama dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Menyiapkan Server Autentikasi/Otorisasi Lokal”
2. the section called “Membuat Proyek Chatterbox”
3. the section called “Hubungkan dengan Ruang Obrolan dan Amati Pembaruan Koneksi”
4. the section called “Membangun Penyedia Token”
5. the section called “Langkah Berikutnya”

Untuk SDK dokumentasi lengkap, mulailah dengan Pesan Klien IVS Obrolan Amazon SDK (di sini di Panduan Pengguna IVS Obrolan Amazon) dan Pesan Klien Obrolan: SDK untuk Referensi Android (aktif GitHub).

Prasyarat

- Kenali Kotlin dan cara membuat aplikasi di platform Android. Jika Anda tidak terbiasa membuat aplikasi untuk Android, pelajari dasar-dasarnya dalam panduan Membangun aplikasi pertama Anda untuk para developer Android.
- Baca dan pahami Memulai dengan IVS Chat dengan saksama.
- Buat AWS IAM pengguna dengan `CreateRoom` kemampuan `CreateChatToken` dan yang ditentukan dalam IAM kebijakan yang ada. (Lihat Memulai dengan IVS Chat).

- Pastikan bahwa kunci rahasia/akses untuk pengguna ini disimpan dalam file AWS kredensial. Untuk petunjuk, lihat Panduan AWS CLI Pengguna (terutama Konfigurasi dan pengaturan file kredenal).
- Buat ruang obrolan dan simpanARN. Lihat Memulai dengan IVS Chat. (Jika Anda tidak menyimpanARN, Anda dapat mencarinya nanti dengan konsol atau ObrolanAPI.)

Menyiapkan Server Autentikasi/Otorisasi Lokal

Server backend Anda akan bertanggung jawab untuk membuat ruang obrolan dan menghasilkan token obrolan yang diperlukan untuk Android SDK Obrolan untuk mengautentikasi dan mengotorisasi klien Anda untuk ruang IVS obrolan Anda.

Lihat Membuat Token Obrolan di Memulai IVS Obrolan Amazon. Seperti yang ditunjukkan pada diagram alur di sana, aplikasi sisi server Anda bertanggung jawab untuk membuat token obrolan. Hal ini berarti aplikasi Anda harus menyediakan caranya sendiri untuk menghasilkan token obrolan dengan memintanya dari aplikasi sisi server Anda.

Kami menggunakan kerangka kerja Ktor untuk membuat server lokal langsung yang mengelola pembuatan token obrolan menggunakan AWS lingkungan lokal Anda.

Pada titik ini, kami berharap AWS kredensialnya sudah diatur dengan benar. Untuk step-by-step petunjuk, lihat Menyiapkan AWS Kredensial dan Wilayah untuk Pengembangan.

Buat direktori baru dan beri nama `chatterbox`, lalu di dalamnya, buat direktori lain yang diberi nama `auth-server`.

Folder server kita akan memiliki struktur sebagai berikut:

```
- auth-server
  - src
    - main
      - kotlin
        - com
          - chatterbox
            - authserver
              - Application.kt
       - resources
         - application.conf
         - logback.xml
   - build.gradle.kts
```

Catatan: Anda dapat langsung menyalin/menempelkan kode di sini ke file yang direferensikan.

Selanjutnya, kita menambahkan semua dependensi dan plugin yang diperlukan agar server autentikasi berfungsi:

Skrip Kotlin:

```
// ./auth-server/build.gradle.kts

plugins {
   application
   kotlin("jvm")
   kotlin("plugin.serialization").version("1.7.10")
}

application {
   mainClass.set("io.ktor.server.netty.EngineMain")
}

dependencies {
   implementation("software.amazon.awssdk:ivschat:2.18.1")
   implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8:1.7.20")

   implementation("io.ktor:ktor-server-core:2.1.3")
   implementation("io.ktor:ktor-server-netty:2.1.3")
   implementation("io.ktor:ktor-server-content-negotiation:2.1.3")
   implementation("io.ktor:ktor-serialization-kotlinx-json:2.1.3")

   implementation("ch.qos.logback:logback-classic:1.4.4")
}
```

Sekarang kita perlu menyiapkan fungsionalitas pembuatan log untuk server autentikasi. (Untuk informasi selengkapnya, lihat Konfigurasikan pembuat log.)

XML:

```
// ./auth-server/src/main/resources/logback.xml

<configuration>
   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <encoder>
         <pattern>%d{YYYY-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</
pattern>
```

```
      </encoder>
   </appender>
   <root level="trace">
      <appender-ref ref="STDOUT"/>
   </root>
   <logger name="org.eclipse.jetty" level="INFO"/>
   <logger name="io.netty" level="INFO"/>
</configuration>
```

Server Ktor memerlukan pengaturan konfigurasi, yang dimuat dari file `application.*` di direktori `resources` secara otomatis, sehingga kita menambahkannya juga. (Untuk informasi selengkapnya, lihat Konfigurasi dalam file.)

HOCON:

```
// ./auth-server/src/main/resources/application.conf

ktor {
   deployment {
      port = 3000
   }
   application {
      modules = [ com.chatterbox.authserver.ApplicationKt.main ]
   }
}
```

Terakhir, mari kita implementasikan server kita:

Kotlin:

```
// ./auth-server/src/main/kotlin/com/chatterbox/authserver/Application.kt

package com.chatterbox.authserver

import io.ktor.http.*
import io.ktor.serialization.kotlinx.json.*
import io.ktor.server.application.*
import io.ktor.server.plugins.contentnegotiation.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import kotlinx.serialization.Serializable
import kotlinx.serialization.json.Json
```

```
import software.amazon.awssdk.services.ivschat.IvschatClient
import software.amazon.awssdk.services.ivschat.model.CreateChatTokenRequest

@Serializable
data class ChatTokenParams(var userId: String, var roomIdentifier: String)

@Serializable
data class ChatToken(
   val token: String,
   val sessionExpirationTime: String,
   val tokenExpirationTime: String,
)

fun Application.main() {
   install(ContentNegotiation) {
      json(Json)
   }

   routing {
      post("/create_chat_token") {
         val callParameters = call.receive<ChatTokenParams>()
         val request =
 CreateChatTokenRequest.builder().roomIdentifier(callParameters.roomIdentifier)
            .userId(callParameters.userId).build()
         val token = IvschatClient.create()
            .createChatToken(request)

         call.respond(
            ChatToken(
                token.token(),
                token.sessionExpirationTime().toString(),
                token.tokenExpirationTime().toString()
            )
         )
      }
   }
}
```

Membuat Proyek Chatterbox

Untuk membuat proyek Android, instal dan buka Android Studio.

Ikuti langkah-langkah yang tercantum dalam Panduan Membuat Proyek Android yang resmi.

- Dalam Pilih tipe proyek Anda, pilih templat proyek Aktivitas Kosong untuk aplikasi Chatterbox kita.
- Di Konfigurasikan proyek Anda, pilih nilai berikut untuk bidang konfigurasi:
  - Nama: Aplikasi Saya
  - Nama paket: com.chatterbox.myapp
  - Simpan lokasi: arahkan ke direktori `chatterbox` yang dibuat di langkah sebelumnya
  - Bahasa: Kotlin
  - APITingkat minimum: API 21: Android 5.0 (Lollipop)

Setelah menentukan semua parameter konfigurasi dengan benar, struktur file kita di dalam folder `chatterbox` akan terlihat seperti berikut:

```
- app
  - build.gradle
  ...
- gradle
- .gitignore
- build.gradle
- gradle.properties
- gradlew
- gradlew.bat
- local.properties
- settings.gradle
- auth-server
  - src
    - main
      - kotlin
        - com
          - chatterbox
            - authserver
              - Application.kt
       - resources
         - application.conf
         - logback.xml
   - build.gradle.kts
```

Sekarang kita memiliki proyek Android yang berfungsi, kita dapat menambahkan com.amazonaws:ivs-chat-messaging ke dependensi kita. `build.gradle` (Untuk informasi selengkapnya tentang kit alat build Gradle, lihat Mengonfigurasi build Anda.)

Catatan: Di bagian atas setiap cuplikan kode, ada jalur ke file tempat Anda harus membuat perubahan dalam proyek. Jalur tersebut bersifat relatif terhadap root proyek.

Dalam kode di bawah ini, ganti *<version>* dengan nomor versi Android Obrolan saat ini SDK (mis., 1.0.0).

Kotlin:

```
// ./app/build.gradle

plugins {
// ...
}

android {
// ...
}

dependencies {
   implementation("com.amazonaws:ivs-chat-messaging:<version>")
// ...
}
```

Setelah dependensi baru ditambahkan, jalankan Sinkronkan Proyek dengan File Gradle di Android Studio untuk menyinkronkan proyek dengan dependensi baru. (Untuk informasi selengkapnya, lihat Menambahkan dependensi build.)

Agar mudah menjalankan server autentikasi (yang dibuat di bagian sebelumnya) dari root proyek, kita memasukkannya sebagai modul baru di `settings.gradle`. (Untuk informasi selengkapnya, lihat Menyusun dan Membangun Komponen Perangkat Lunak dengan Gradle.)

Skrip Kotlin:

```
// ./settings.gradle

// ...

rootProject.name = "Chatterbox"
include ':app'
include ':auth-server'
```

Mulai sekarang, karena `auth-server` disertakan dalam proyek Android, Anda dapat menjalankan server autentikasi dengan perintah berikut dari root proyek:

Shell:

```
./gradlew :auth-server:run
```

Hubungkan dengan Ruang Obrolan dan Amati Pembaruan Koneksi

Untuk membuka koneksi ruang obrolan, kami menggunakan callback siklus hidup aktivitas onCreate (), yang diaktifkan saat aktivitas pertama kali dibuat. ChatRoom Konstruktor mengharuskan kami untuk menyediakan `region` dan `tokenProvider` membuat instance koneksi ruangan.

Catatan: Fungsi `fetchChatToken` dalam cuplikan di bawah ini akan diimplementasikan di bagian berikutnya.

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp

// ...
import androidx.appcompat.app.AppCompatActivity
// ...

// AWS region of the room that was created in Getting Started with Amazon IVS Chat
const val REGION = "us-west-2"

class MainActivity : AppCompatActivity() {
    private var room: ChatRoom? = null
    // ...

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)

      // Create room instance
      room = ChatRoom(REGION, ::fetchChatToken)
   }
```

```
// ...
}
```

Menampilkan dan bereaksi terhadap perubahan dalam koneksi ruang obrolan adalah bagian penting dari pembuatan aplikasi obrolan, seperti `chatterbox`. Sebelum kita dapat mulai berinteraksi dengan ruang tersebut, kita harus berlangganan peristiwa status koneksi ruang obrolan untuk mendapatkan pembaruan.

ChatRoommengharapkan kami untuk melampirkan implementasi ChatRoomListener antarmuka untuk meningkatkan peristiwa siklus hidup. Untuk saat ini, fungsi pendengar hanya akan membuat log pesan konfirmasi saat diinvokasi:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

// ...
package com.chatterbox.myapp
// ...
const val TAG = "IVSChat-App"

class MainActivity : AppCompatActivity() {
// ...

    private val roomListener = object : ChatRoomListener {
        override fun onConnecting(room: ChatRoom) {
            Log.d(TAG, "onConnecting")
        }

        override fun onConnected(room: ChatRoom) {
            Log.d(TAG, "onConnected")
        }

        override fun onDisconnected(room: ChatRoom, reason: DisconnectReason) {
            Log.d(TAG, "onDisconnected $reason")
        }

        override fun onMessageReceived(room: ChatRoom, message: ChatMessage) {
            Log.d(TAG, "onMessageReceived $message")
        }

        override fun onMessageDeleted(room: ChatRoom, event: DeleteMessageEvent) {
            Log.d(TAG, "onMessageDeleted $event")
```

```
        }

        override fun onEventReceived(room: ChatRoom, event: ChatEvent) {
            Log.d(TAG, "onEventReceived $event")
        }

        override fun onUserDisconnected(room: ChatRoom, event:    DisconnectUserEvent)
 {
            Log.d(TAG, "onUserDisconnected $event")
        }
    }
}
```

Setelah menerapkan `ChatRoomListener`, kita melampirkannya ke instans ruang:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   binding = ActivityMainBinding.inflate(layoutInflater)
   setContentView(binding.root)

   // Create room instance
   room = ChatRoom(REGION, ::fetchChatToken).apply {
      listener = roomListener
   }
}

private val roomListener = object : ChatRoomListener {
// ...
}
```

Setelah ini, kita perlu menyediakan kemampuan untuk membaca status koneksi ruang. Kami akan menyimpannya di `MainActivity.kt` properti dan menginisialisasikannya ke DISCONNECTED status default untuk kamar (lihat `ChatRoom state` di SDKReferensi Android IVS Obrolan). Agar tetap dapat memperbarui status lokal, kita perlu mengimplementasikan fungsi state-updater; sebut saja `updateConnectionState`:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

enum class ConnectionState {
   CONNECTED,
   DISCONNECTED,
   LOADING
}

class MainActivity : AppCompatActivity() {
   private var connectionState = ConnectionState.DISCONNECTED
// ...

   private fun updateConnectionState(state: ConnectionState) {
      connectionState = state

      when (state) {
         ConnectionState.CONNECTED -> {
            Log.d(TAG, "room connected")
         }
         ConnectionState.DISCONNECTED -> {
            Log.d(TAG, "room disconnected")
         }
         ConnectionState.LOADING -> {
            Log.d(TAG, "room loading")
         }
      }
   }
}
```

Selanjutnya, kita mengintegrasikan fungsi state-updater kita dengan properti.listener: ChatRoom

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...
```

```
class MainActivity : AppCompatActivity() {
// ...

   private val roomListener = object : ChatRoomListener {
      override fun onConnecting(room: ChatRoom) {
         Log.d(TAG, "onConnecting")
         runOnUiThread {
            updateConnectionState(ConnectionState.LOADING)
         }
      }

      override fun onConnected(room: ChatRoom) {
         Log.d(TAG, "onConnected")
         runOnUiThread {
            updateConnectionState(ConnectionState.CONNECTED)
         }
      }

      override fun onDisconnected(room: ChatRoom, reason: DisconnectReason) {
         Log.d(TAG, "[${Thread.currentThread().name}] onDisconnected")
         runOnUiThread {
            updateConnectionState(ConnectionState.DISCONNECTED)
         }
      }
   }
}
```

Sekarang kita memiliki kemampuan untuk menyimpan, mendengarkan, dan bereaksi terhadap pembaruan ChatRoomstatus, saatnya untuk menginisialisasi koneksi:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

enum class ConnectionState {
   CONNECTED,
   DISCONNECTED,
   LOADING
}
```

```
class MainActivity : AppCompatActivity() {
   private var connectionState = ConnectionState.DISCONNECTED
// ...

   private fun connect() {
      try {
         room?.connect()
      } catch (ex: Exception) {
         Log.e(TAG, "Error while calling connect()", ex)
      }
   }

   private val roomListener = object : ChatRoomListener {
      // ...
      override fun onConnecting(room: ChatRoom) {
         Log.d(TAG, "onConnecting")
         runOnUiThread {
            updateConnectionState(ConnectionState.LOADING)
         }
      }

      override fun onConnected(room: ChatRoom) {
         Log.d(TAG, "onConnected")
         runOnUiThread {
            updateConnectionState(ConnectionState.CONNECTED)
         }
      }
      // ...
   }
}
```

Membangun Penyedia Token

Kini saatnya membuat fungsi yang bertanggung jawab untuk membuat dan mengelola token obrolan di aplikasi. Dalam contoh ini kami menggunakan HTTPklien Retrofit untuk Android.

Sebelum kita dapat mengirim lalu lintas jaringan yang ada, kita harus menyiapkan konfigurasi keamanan jaringan untuk Android. (Untuk informasi selengkapnya, lihat Konfigurasikan keamanan jaringan.) Kita mulai dengan menambahkan izin jaringan ke file Manifes Aplikasi. Perhatikan tanda `user-permission` dan atribut `networkSecurityConfig` yang ditambahkan, yang akan mengarahkan ke konfigurasi keamanan jaringan baru kita. Dalam kode di bawah ini, ganti *<version>* dengan nomor versi Android Obrolan saat ini SDK (mis., 1.0.0).

XML:

```
// ./app/src/main/AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.chatterbox.myapp">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:fullBackupContent="@xml/backup_rules"
        android:label="@string/app_name"
        android:networkSecurityConfig="@xml/network_security_config"
// ...

// ./app/build.gradle


dependencies {
   implementation("com.amazonaws:ivs-chat-messaging:<version>")
// ...

   implementation("com.squareup.retrofit2:retrofit:2.9.0")
}
```

Nyatakan domain `10.0.2.2` dan `localhost` sebagai tepercaya, untuk mulai bertukar pesan dengan backend kami:

XML:

```
// ./app/src/main/res/xml/network_security_config.xml

<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">10.0.2.2</domain>
        <domain includeSubdomains="true">localhost</domain>
    </domain-config>
</network-security-config>
```

Selanjutnya, kita perlu menambahkan dependensi baru, bersama dengan penambahan konverter Gson untuk respons HTTP parsing. Dalam kode di bawah ini, ganti *<version>* dengan nomor versi Android Obrolan saat ini SDK (mis., 1.0.0).

Skrip Kotlin:

```
// ./app/build.gradle

dependencies {
   implementation("com.amazonaws:ivs-chat-messaging:<version>")
// ...

   implementation("com.squareup.retrofit2:retrofit:2.9.0")
}
```

Untuk mengambil token obrolan, kita perlu membuat POST HTTP permintaan dari `chatterbox` aplikasi kita. Kita menentukan permintaan dalam antarmuka Retrofit yang akan diimplementasikan. (Lihat dokumentasi Retrofit. Juga biasakan diri Anda dengan spesifikasi CreateChatTokentitik akhir.)

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/network/ApiService.kt

package com.chatterbox.myapp.network
// ...


import androidx.annotation.Keep
import com.amazonaws.ivs.chat.messaging.ChatToken
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.POST

data class CreateTokenParams(var userId: String, var roomIdentifier: String)

interface ApiService {
   @POST("create_chat_token")
   fun createChatToken(@Body params: CreateTokenParams): Call<ChatToken>
}
```

Dengan jaringan yang telah siap, kini saatnya menambahkan fungsi yang bertanggung jawab untuk membuat dan mengelola token obrolan kita. Kita menambahkannya ke `MainActivity.kt`, yang secara otomatis dibuat ketika proyek dibuat:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt


package com.chatterbox.myapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import com.amazonaws.ivs.chat.messaging.*
import com.chatterbox.myapp.network.CreateTokenParams
import com.chatterbox.myapp.network.RetrofitFactory
import retrofit2.Call
import java.io.IOException
import retrofit2.Callback
import retrofit2.Response

// custom tag for logging purposes
const val TAG = "IVSChat-App"

// any ID to be associated with auth token
const val USER_ID = "test user id"
// ID of the room the app wants to access. Must be an ARN. See Amazon Resource
 Names(ARNs)
const val ROOM_ID = "arn:aws:..."
// AWS region of the room that was created in Getting Started with Amazon IVS Chat
const val REGION = "us-west-2"

class MainActivity : AppCompatActivity() {
   private val service = RetrofitFactory.makeRetrofitService()
   private lateinit var userId: String

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
   }

   private fun fetchChatToken(callback: ChatTokenCallback) {
```

```
      val params = CreateTokenParams(userId, ROOM_ID)
      service.createChatToken(params).enqueue(object : Callback<ChatToken> {
         override fun onResponse(call: Call<ChatToken>, response: Response<ChatToken>)
 {
            val token = response.body()
            if (token == null) {
               Log.e(TAG, "Received empty token response")
               callback.onFailure(IOException("Empty token response"))
               return
            }

            Log.d(TAG, "Received token response $token")
            callback.onSuccess(token)
         }

         override fun onFailure(call: Call<ChatToken>, throwable: Throwable) {
            Log.e(TAG, "Failed to fetch token", throwable)
            callback.onFailure(throwable)
         }
      })
   }
}
```

Langkah Berikutnya

Setelah Anda membuat koneksi ruang obrolan, lanjutkan ke Bagian 2 dari tutorial Android ini, Pesan dan Peristiwa.

IVSPesan Klien ObrolanSDK: Tutorial Android Bagian 2: Pesan dan Acara

Bagian kedua (dan terakhir) dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Membuat UI untuk Mengirim Pesan”
   a. the section called “Tata Letak Utama UI”
   b. the section called “Sel Teks Abstraksi UI untuk Menampilkan Teks Secara Konsisten”
   c. the section called “Pesan Obrolan Kiri UI”
   d. the section called “Pesan Obrolan Kanan UI”
   e. the section called “Nilai Warna Tambahan UI”

2. the section called “Menerapkan Ikatan Tampilan”
3. the section called “Mengelola Permintaan Pesan-Obrolan”
4. the section called “Langkah Terakhir”

Untuk SDK dokumentasi lengkap, mulailah dengan Pesan Klien IVS Obrolan Amazon SDK (di sini di Panduan Pengguna IVS Obrolan Amazon) dan Pesan Klien Obrolan: SDK untuk Referensi Android (aktif GitHub).

Prasyarat

Pastikan Anda telah menyelesaikan Bagian 1 dari tutorial ini, Ruang Obrolan.

Membuat UI untuk Mengirim Pesan

Setelah kita berhasil menginisialisasi koneksi ruang obrolan, kini saatnya mengirim pesan pertama kami. UI diperlukan untuk fitur ini. Kami akan menambahkan:

- Tombol `connect/disconnect`
- Input pesan dengan tombol `send`
- Daftar pesan dinamis. Untuk membangun ini, kami menggunakan Android Jetpack RecyclerView.

Tata Letak Utama UI

Lihat Tata Letak Android Jetpack di dokumentasi developer Android.

XML:

```
// ./app/src/main/res/layout/activity_main.xml


<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://
schemas.android.com/apk/res/android"
                                                     xmlns:app="http://
schemas.android.com/apk/res-auto"
                                                     xmlns:tools="http://
schemas.android.com/tools"

 android:layout_width="match_parent"
```

```
android:layout_height="match_parent">

   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
                 xmlns:app="http://schemas.android.com/apk/res-auto"
                 android:id="@+id/connect_view"
                 android:layout_width="match_parent"
                 android:layout_height="match_parent"
                 android:gravity="center"
                 android:orientation="vertical">

       <androidx.cardview.widget.CardView
               android:id="@+id/connect_button"
               android:layout_width="match_parent"
               android:layout_height="48dp"
               android:layout_gravity=""
               android:layout_marginStart="16dp"
               android:layout_marginTop="4dp"
               android:layout_marginEnd="16dp"
               android:clickable="true"
               android:elevation="16dp"
               android:focusable="true"
               android:foreground="?android:attr/selectableItemBackground"
               app:cardBackgroundColor="@color/purple_500"
               app:cardCornerRadius="10dp">

           <TextView
                   android:id="@+id/connect_text"
                   android:layout_width="wrap_content"
                   android:layout_height="wrap_content"
                   android:layout_alignParentEnd="true"
                   android:layout_gravity="center"
                   android:layout_weight="1"
                   android:paddingHorizontal="12dp"
                   android:text="Connect"
                   android:textColor="@color/white"
                   android:textSize="16sp"/>

           <ProgressBar
                   android:id="@+id/activity_indicator"
                   android:layout_width="20dp"
                   android:layout_height="20dp"
                   android:layout_gravity="center"
                   android:layout_marginHorizontal="20dp"
```

```
                android:indeterminateOnly="true"
                android:indeterminateTint="@color/white"
                android:indeterminateTintMode="src_atop"
                android:keepScreenOn="true"
                android:visibility="gone"/>
        </androidx.cardview.widget.CardView>

    </LinearLayout>

    <androidx.constraintlayout.widget.ConstraintLayout
            android:id="@+id/chat_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:clipToPadding="false"
            android:visibility="visible"
            tools:context=".MainActivity">

        <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                app:layout_constraintBottom_toTopOf="@+id/layout_message_input"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent">

            <androidx.recyclerview.widget.RecyclerView
                    android:id="@+id/recycler_view"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:clipToPadding="false"
                    android:paddingTop="70dp"
                    android:paddingBottom="20dp"/>
        </RelativeLayout>

        <RelativeLayout
                android:id="@+id/layout_message_input"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@android:color/white"
                android:clipToPadding="false"
                android:drawableTop="@android:color/black"
                android:elevation="18dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent">
```

```
            <EditText
                    android:id="@+id/message_edit_text"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_centerVertical="true"
                    android:layout_marginStart="16dp"
                    android:layout_toStartOf="@+id/send_button"
                    android:background="@android:color/transparent"
                    android:hint="Enter Message"
                    android:inputType="text"
                    android:maxLines="6"
                    tools:ignore="Autofill"/>

            <Button
                    android:id="@+id/send_button"
                    android:layout_width="84dp"
                    android:layout_height="48dp"
                    android:layout_alignParentEnd="true"
                    android:background="@color/black"
                    android:foreground="?android:attr/selectableItemBackground"
                    android:text="Send"
                    android:textColor="@color/white"
                    android:textSize="12dp"/>
        </RelativeLayout>
    </androidx.constraintlayout.widget.ConstraintLayout>


</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Sel Teks Abstraksi UI untuk Menampilkan Teks Secara Konsisten

XML:

```
// ./app/src/main/res/layout/common_cell.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/layout_container"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:background="@color/light_gray"
```

```
            android:minWidth="100dp"
            android:orientation="vertical">

    <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal">

        <TextView
                android:id="@+id/card_message_me_text_view"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_marginBottom="8dp"
                android:maxWidth="260dp"
                android:paddingLeft="12dp"
                android:paddingTop="8dp"
                android:paddingRight="12dp"
                android:text="This is a Message"
                android:textColor="#ffffff"
                android:textSize="16sp"/>

        <TextView
                android:id="@+id/failed_mark"
                android:layout_width="40dp"
                android:layout_height="match_parent"
                android:paddingRight="5dp"
                android:src="@drawable/ic_launcher_background"
                android:text="!"
                android:textAlignment="viewEnd"
                android:textColor="@color/white"
                android:textSize="25dp"
                android:visibility="gone"/>
    </LinearLayout>

</LinearLayout>
```

Pesan Obrolan Kiri UI

XML:

```
// ./app/src/main/res/layout/card_view_left.xml

<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:app="http://schemas.android.com/apk/res-auto"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_marginStart="8dp"
              android:layout_marginBottom="12dp"
              android:orientation="vertical">

    <TextView
            android:id="@+id/username_edit_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="UserName"/>

    <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

        <androidx.cardview.widget.CardView
                android:id="@+id/card_message_other"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="left"
                android:layout_marginBottom="4dp"
                android:foreground="?android:attr/selectableItemBackground"
                app:cardBackgroundColor="@color/light_gray_2"
                app:cardCornerRadius="10dp"
                app:cardElevation="0dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent">

            <include layout="@layout/common_cell"/>
        </androidx.cardview.widget.CardView>

        <TextView
                android:id="@+id/dateText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="4dp"
                android:layout_marginBottom="4dp"
                android:text="10:00"
                app:layout_constraintBottom_toBottomOf="@+id/card_message_other"
                app:layout_constraintLeft_toRightOf="@+id/card_message_other"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

```
</LinearLayout>
```

Pesan Obrolan Kanan UI

XML:

```
// ./app/src/main/res/layout/card_view_right.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://
schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
                android:layout_width="match_parent"
               android:layout_height="wrap_content"
android:layout_marginEnd="8dp">

    <androidx.cardview.widget.CardView
            android:id="@+id/card_message_me"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:layout_marginBottom="10dp"
            android:foreground="?android:attr/selectableItemBackground"
            app:cardBackgroundColor="@color/purple_500"
            app:cardCornerRadius="10dp"
            app:cardElevation="0dp"
            app:cardPreventCornerOverlap="false"
            app:cardUseCompatPadding="true"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent">

        <include layout="@layout/common_cell"/>

    </androidx.cardview.widget.CardView>

    <TextView
            android:id="@+id/dateText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginRight="12dp"
```

```
            android:layout_marginBottom="4dp"
            android:text="10:00"
            app:layout_constraintBottom_toBottomOf="@+id/card_message_me"
            app:layout_constraintRight_toLeftOf="@+id/card_message_me"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Nilai Warna Tambahan UI

XML:

```
// ./app/src/main/res/values/colors.xml

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--    ...-->
    <color name="dark_gray">#4F4F4F</color>
    <color name="blue">#186ED3</color>
    <color name="dark_red">#b30000</color>
    <color name="light_gray">#B7B7B7</color>
    <color name="light_gray_2">#eef1f6</color>
</resources>
```

Menerapkan Ikatan Tampilan

Kami memanfaatkan fitur Android View Binding untuk dapat mereferensikan class binding untuk XML tata letak kami. Untuk mengaktifkan fitur tersebut, atur opsi build `viewBinding` ke `true` pada `./app/build.gradle`:

Skrip Kotlin:

```
 // ./app/build.gradle

android {
//    ...

    buildFeatures {
        viewBinding = true
    }
//    ...
}
```

Sekarang saatnya menghubungkan UI dengan kode Kotlin:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt
package com.chatterbox.myapp
// ...
const val TAG = "Chatterbox-MyApp"


class MainActivity : AppCompatActivity() {
//    ...

    private fun sendMessage(request: SendMessageRequest) {
        try {
            room?.sendMessage(
                request,
                object : SendMessageCallback {
                    override fun onRejected(request: SendMessageRequest, error:
 ChatError) {
                        runOnUiThread {
                            entries.addFailedRequest(request)
                            scrollToBottom()
                            Log.e(TAG, "Message rejected: ${error.errorMessage}")
                        }
                    }
                }
            )

            entries.addPendingRequest(request)

            binding.messageEditText.text.clear()
            scrollToBottom()
        } catch (error: Exception) {
            Log.e(TAG, error.message ?: "Unknown error occurred")
        }
    }

    private fun scrollToBottom() {
        binding.recyclerView.smoothScrollToPosition(entries.size - 1)
    }

    private fun sendButtonClick(view: View) {
        val content = binding.messageEditText.text.toString()
```

```
        if (content.trim().isEmpty()) {
            return
        }

        val request = SendMessageRequest(content)
        sendMessage(request)
    }
}
```

Kita juga menambahkan metode untuk menghapus pesan dan memutus koneksi pengguna dari obrolan, yang dapat diinvokasi menggunakan menu konteks pesan obrolan:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
//    ...

class MainActivity : AppCompatActivity() {
//    ...

    private fun deleteMessage(request: DeleteMessageRequest) {
        room?.deleteMessage(
            request,
            object : DeleteMessageCallback {
                override fun onRejected(request: DeleteMessageRequest, error:
 ChatError) {
                    runOnUiThread {
                        Log.d(TAG, "Delete message rejected: ${error.errorMessage}")
                    }
                }
            }
        )
    }

    private fun disconnectUser(request: DisconnectUserRequest) {
        room?.disconnectUser(
            request,
            object : DisconnectUserCallback {
                override fun onRejected(request: DisconnectUserRequest, error:
 ChatError) {
                    runOnUiThread {
```

```
                        Log.d(TAG, "Disconnect user rejected: ${error.errorMessage}")
                    }
                }
            }
        )
    }
}
```

Mengelola Permintaan Pesan-Obrolan

Kami membutuhkan cara untuk mengelola permintaan pesan-obrolan melalui semua kemungkinan statusnya:

- Tertunda — Pesan telah dikirim ke ruang obrolan, tetapi belum dikonfirmasi atau ditolak.
- Dikonfirmasi — Pesan dikirim oleh ruang obrolan ke semua pengguna (termasuk kita).
- Ditolak — Pesan ditolak oleh ruang obrolan dengan objek kesalahan.

Kita akan menyimpan permintaan obrolan dan pesan obrolan yang belum terselesaikan dalam suatu daftar. Daftar ini layak mendapat kelas terpisah, yang kita sebut `ChatEntries.kt`:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/ChatEntries.kt

package com.chatterbox.myapp

import com.amazonaws.ivs.chat.messaging.entities.ChatMessage
import com.amazonaws.ivs.chat.messaging.requests.SendMessageRequest

sealed class ChatEntry() {
    class Message(val message: ChatMessage) : ChatEntry()
    class PendingRequest(val request: SendMessageRequest) : ChatEntry()
    class FailedRequest(val request: SendMessageRequest) : ChatEntry()
}

class ChatEntries {
    /* This list is kept in sorted order. ChatMessages are sorted by date, while
 pending and failed requests are kept in their original insertion point. */
    val entries = mutableListOf<ChatEntry>()
    var adapter: ChatListAdapter? = null
```

```
    val size get() = entries.size

    /**
     * Insert pending request at the end.
     */
    fun addPendingRequest(request: SendMessageRequest) {
        val insertIndex = entries.size
        entries.add(insertIndex, ChatEntry.PendingRequest(request))
        adapter?.notifyItemInserted(insertIndex)
    }

    /**
     * Insert received message at proper place based on sendTime. This can cause
removal of pending requests.
     */
    fun addReceivedMessage(message: ChatMessage) {
        /* Skip if we have already handled that message. */
        val existingIndex = entries.indexOfLast { it is ChatEntry.Message &&
it.message.id == message.id }
        if (existingIndex != -1) {
            return
        }

        val removeIndex = entries.indexOfLast {
            it is ChatEntry.PendingRequest && it.request.requestId == message.requestId
        }
        if (removeIndex != -1) {
            entries.removeAt(removeIndex)
        }

        val insertIndexRaw = entries.indexOfFirst { it is ChatEntry.Message &&
it.message.sendTime > message.sendTime }
        val insertIndex = if (insertIndexRaw == -1) entries.size else insertIndexRaw
        entries.add(insertIndex, ChatEntry.Message(message))

        if (removeIndex == -1) {
            adapter?.notifyItemInserted(insertIndex)
        } else if (removeIndex == insertIndex) {
            adapter?.notifyItemChanged(insertIndex)
        } else {
            adapter?.notifyItemRemoved(removeIndex)
            adapter?.notifyItemInserted(insertIndex)
        }
    }
```

```
    fun addFailedRequest(request: SendMessageRequest) {
        val removeIndex = entries.indexOfLast {
            it is ChatEntry.PendingRequest && it.request.requestId == request.requestId
        }
        if (removeIndex != -1) {
            entries.removeAt(removeIndex)
            entries.add(removeIndex, ChatEntry.FailedRequest(request))
            adapter?.notifyItemChanged(removeIndex)
        } else {
            val insertIndex = entries.size
            entries.add(insertIndex, ChatEntry.FailedRequest(request))
            adapter?.notifyItemInserted(insertIndex)
        }
    }

    fun removeMessage(messageId: String) {
        val removeIndex = entries.indexOfFirst { it is ChatEntry.Message &&
 it.message.id == messageId }
        entries.removeAt(removeIndex)
        adapter?.notifyItemRemoved(removeIndex)
    }

    fun removeFailedRequest(requestId: String) {
        val removeIndex = entries.indexOfFirst { it is ChatEntry.FailedRequest &&
 it.request.requestId == requestId }
        entries.removeAt(removeIndex)
        adapter?.notifyItemRemoved(removeIndex)
    }

    fun removeAll() {
        entries.clear()
    }
}
```

Untuk menghubungkan daftar dengan UI, kita menggunakan Adaptor. Untuk informasi selengkapnya, lihat Mengikat Data dengan AdapterView dan kelas pengikatan yang dihasilkan.

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/ChatListAdapter.kt

package com.chatterbox.myapp
```

```
import android.content.Context
import android.graphics.Color
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.LinearLayout
import android.widget.TextView
import androidx.core.content.ContextCompat
import androidx.core.view.isGone
import androidx.recyclerview.widget.RecyclerView
import com.amazonaws.ivs.chat.messaging.requests.DisconnectUserRequest
import java.text.DateFormat


class ChatListAdapter(
    private val entries: ChatEntries,
    private val onDisconnectUser: (request: DisconnectUserRequest) -> Unit,
) :
    RecyclerView.Adapter<ChatListAdapter.ViewHolder>() {
    var context: Context? = null
    var userId: String? = null

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val container: LinearLayout = view.findViewById(R.id.layout_container)
        val textView: TextView = view.findViewById(R.id.card_message_me_text_view)
        val failedMark: TextView = view.findViewById(R.id.failed_mark)
        val userNameText: TextView? = view.findViewById(R.id.username_edit_text)
        val dateText: TextView? = view.findViewById(R.id.dateText)
    }

    override fun onCreateViewHolder(viewGroup: ViewGroup, viewType: Int): ViewHolder {
        if (viewType == 0) {
            val rightView =
 LayoutInflater.from(viewGroup.context).inflate(R.layout.card_view_right, viewGroup,
 false)
            return ViewHolder(rightView)
        }
        val leftView =
 LayoutInflater.from(viewGroup.context).inflate(R.layout.card_view_left, viewGroup,
 false)
        return ViewHolder(leftView)
    }
```

```
    override fun getItemViewType(position: Int): Int {
        // Int 0 indicates to my message while Int 1 to other message
        val chatMessage = entries.entries[position]
        return if (chatMessage is ChatEntry.Message &&
chatMessage.message.sender.userId != userId) 1 else 0
    }

    override fun onBindViewHolder(viewHolder: ViewHolder, position: Int) {
        return when (val entry = entries.entries[position]) {
            is ChatEntry.Message -> {
                viewHolder.textView.text = entry.message.content

                val bgColor = if (entry.message.sender.userId == userId) {
                    R.color.purple_500
                } else {
                    R.color.light_gray_2
                }

viewHolder.container.setBackgroundColor(ContextCompat.getColor(context!!, bgColor))

                if (entry.message.sender.userId != userId) {
                    viewHolder.textView.setTextColor(Color.parseColor("#000000"))
                }

                viewHolder.failedMark.isGone = true

                viewHolder.itemView.setOnCreateContextMenuListener { menu, _, _ ->
                    menu.add("Kick out").setOnMenuItemClickListener {
                        val request =
DisconnectUserRequest(entry.message.sender.userId, "Some reason")
                        onDisconnectUser(request)
                        true
                    }
                }

                viewHolder.userNameText?.text = entry.message.sender.userId
                viewHolder.dateText?.text =

DateFormat.getTimeInstance(DateFormat.SHORT).format(entry.message.sendTime)
            }

            is ChatEntry.PendingRequest -> {
```

```
 viewHolder.container.setBackgroundColor(ContextCompat.getColor(context!!,
 R.color.light_gray))
                viewHolder.textView.text = entry.request.content
                viewHolder.failedMark.isGone = true
                viewHolder.itemView.setOnCreateContextMenuListener(null)
                viewHolder.dateText?.text = "Sending"
            }

            is ChatEntry.FailedRequest -> {
                viewHolder.textView.text = entry.request.content

 viewHolder.container.setBackgroundColor(ContextCompat.getColor(context!!,
 R.color.dark_red))
                viewHolder.failedMark.isGone = false
                viewHolder.dateText?.text = "Failed"
            }
        }
    }

    override fun onAttachedToRecyclerView(recyclerView: RecyclerView) {
        super.onAttachedToRecyclerView(recyclerView)
        context = recyclerView.context
    }

    override fun getItemCount() = entries.entries.size
}
```

Langkah Terakhir

Saatnya menghubungkan adaptor baru kita, dengan mengikat kelas `ChatEntries` ke `MainActivity`:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

import com.chatterbox.myapp.databinding.ActivityMainBinding
import com.chatterbox.myapp.ChatListAdapter
import com.chatterbox.myapp.ChatEntries
```

```
class MainActivity : AppCompatActivity() {
    // ...
    private var entries = ChatEntries()
    private lateinit var adapter: ChatListAdapter
    private lateinit var binding: ActivityMainBinding

    /* see https://developer.android.com/topic/libraries/data-binding/generated-
binding#create */
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        /* Create room instance. */
        room = ChatRoom(REGION, ::fetchChatToken).apply {
            listener = roomListener
        }

        binding.sendButton.setOnClickListener(::sendButtonClick)
        binding.connectButton.setOnClickListener { connect() }

        setUpChatView()

        updateConnectionState(ConnectionState.DISCONNECTED)
    }

    private fun setUpChatView() {
        /* Setup Android Jetpack RecyclerView - see https://developer.android.com/
develop/ui/views/layout/recyclerview.*/
        adapter = ChatListAdapter(entries, ::disconnectUser)
        entries.adapter = adapter

        val recyclerViewLayoutManager = LinearLayoutManager(this@MainActivity,
 LinearLayoutManager.VERTICAL, false)
        binding.recyclerView.layoutManager = recyclerViewLayoutManager
        binding.recyclerView.adapter = adapter

        binding.sendButton.setOnClickListener(::sendButtonClick)
        binding.messageEditText.setOnEditorActionListener { _, _, event ->
            val isEnterDown = (event.action == KeyEvent.ACTION_DOWN) && (event.keyCode
 == KeyEvent.KEYCODE_ENTER)
            if (!isEnterDown) {
                return@setOnEditorActionListener false
```

```
            }

            sendButtonClick(binding.sendButton)
            return@setOnEditorActionListener true
        }
    }
}
```

Karena kita sudah memiliki kelas yang bertanggung jawab untuk melacak permintaan obrolan (`ChatEntries`), kita siap mengimplementasikan kode untuk memanipulasi `entries` di `roomListener`. Kita akan memperbarui `entries` dan `connectionState` sesuai dengan peristiwa yang kita tanggapi:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

class MainActivity : AppCompatActivity() {
    //...


    private fun sendMessage(request: SendMessageRequest) {
    //...

    }

    private fun scrollToBottom() {
        binding.recyclerView.smoothScrollToPosition(entries.size - 1)
    }

    private val roomListener = object : ChatRoomListener {
        override fun onConnecting(room: ChatRoom) {
            Log.d(TAG, "[${Thread.currentThread().name}] onConnecting")
            runOnUiThread {
                updateConnectionState(ConnectionState.LOADING)
            }
        }

        override fun onConnected(room: ChatRoom) {
            Log.d(TAG, "[${Thread.currentThread().name}] onConnected")
```

```
            runOnUiThread {
                updateConnectionState(ConnectionState.CONNECTED)
            }
        }

        override fun onDisconnected(room: ChatRoom, reason: DisconnectReason) {
            Log.d(TAG, "[${Thread.currentThread().name}] onDisconnected")
            runOnUiThread {
                updateConnectionState(ConnectionState.DISCONNECTED)
                entries.removeAll()
            }
        }

        override fun onMessageReceived(room: ChatRoom, message: ChatMessage) {
            Log.d(TAG, "[${Thread.currentThread().name}] onMessageReceived $message")
            runOnUiThread {
                entries.addReceivedMessage(message)
                scrollToBottom()
            }
        }

        override fun onEventReceived(room: ChatRoom, event: ChatEvent) {
            Log.d(TAG, "[${Thread.currentThread().name}] onEventReceived $event")
        }

        override fun onMessageDeleted(room: ChatRoom, event: DeleteMessageEvent) {
            Log.d(TAG, "[${Thread.currentThread().name}] onMessageDeleted $event")
        }

        override fun onUserDisconnected(room: ChatRoom, event: DisconnectUserEvent) {
            Log.d(TAG, "[${Thread.currentThread().name}] onUserDisconnected $event")
        }
    }
}
```

Sekarang Anda seharusnya dapat menjalankan aplikasi! (Lihat Membangun dan menjalankan aplikasi Anda.) Ingatlah untuk menjalankan server backend Anda saat menggunakan aplikasi. Anda dapat memutarnya dari terminal di root proyek kita dengan perintah ini: `./gradlew :auth-server:run` atau dengan menjalankan tugas Gradle `auth-server:run` langsung dari Android Studio.

IVSPesan Klien ObrolanSDK: Tutorial Kotlin Coroutines Bagian 1: Ruang Obrolan

Ini adalah bagian pertama dari tutorial dua bagian. Anda akan mempelajari hal-hal penting dalam bekerja dengan Amazon IVS Chat Messaging SDK dengan membangun aplikasi Android yang berfungsi penuh menggunakan bahasa pemrograman Kotlin dan coroutine. Kami menyebut aplikasi itu Chatterbox.

Sebelum Anda memulai modul, luangkan beberapa menit untuk membiasakan diri dengan prasyarat, konsep utama di balik token obrolan, dan server backend yang diperlukan untuk membuat ruang obrolan.

Tutorial ini dibuat untuk pengembang Android berpengalaman yang baru mengenal IVS Chat MessagingSDK. Anda harus merasa nyaman dengan bahasa pemrograman Kotlin dan membuat UIs di platform Android.

Bagian pertama dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Menyiapkan Server Autentikasi/Otorisasi Lokal”
2. the section called “Membuat Proyek Chatterbox”
3. the section called “Hubungkan dengan Ruang Obrolan dan Amati Pembaruan Koneksi”
4. the section called “Membangun Penyedia Token”
5. the section called “Langkah Berikutnya”

Untuk SDK dokumentasi lengkap, mulailah dengan Pesan Klien IVS Obrolan Amazon SDK (di sini di Panduan Pengguna IVS Obrolan Amazon) dan Pesan Klien Obrolan: SDK untuk Referensi Android (aktif GitHub).

Prasyarat

- Kenali Kotlin dan cara membuat aplikasi di platform Android. Jika Anda tidak terbiasa membuat aplikasi untuk Android, pelajari dasar-dasarnya dalam panduan Membangun aplikasi pertama Anda untuk para developer Android.
- Baca dan pahami Memulai dengan IVS Chat.
- Buat AWS IAM pengguna dengan `CreateRoom` kemampuan `CreateChatToken` dan yang ditentukan dalam IAM kebijakan yang ada. (Lihat Memulai dengan IVS Chat).

- Pastikan kunci rahasia/akses untuk pengguna ini disimpan dalam file AWS kredensil. Untuk petunjuk, lihat Panduan AWS CLI Pengguna (terutama Konfigurasi dan pengaturan file kredenal).
- Buat ruang obrolan dan simpanARN. Lihat Memulai dengan IVS Chat. (Jika Anda tidak menyimpanARN, Anda dapat mencarinya nanti dengan konsol atau ObrolanAPI.)

Menyiapkan Server Autentikasi/Otorisasi Lokal

Server backend Anda akan bertanggung jawab untuk membuat ruang obrolan dan menghasilkan token obrolan yang diperlukan untuk Android SDK Obrolan untuk mengautentikasi dan mengotorisasi klien Anda untuk ruang IVS obrolan Anda.

Lihat Membuat Token Obrolan di Memulai IVS Obrolan Amazon. Seperti yang ditunjukkan pada diagram alur di sana, aplikasi sisi server Anda bertanggung jawab untuk membuat token obrolan. Hal ini berarti aplikasi Anda harus menyediakan caranya sendiri untuk menghasilkan token obrolan dengan memintanya dari aplikasi sisi server Anda.

Kami menggunakan kerangka kerja Ktor untuk membuat server lokal langsung yang mengelola pembuatan token obrolan menggunakan AWS lingkungan lokal Anda.

Pada titik ini, kami berharap AWS kredensialnya sudah diatur dengan benar. Untuk step-by-step petunjuk, lihat Menyiapkan kredensial AWS sementara dan AWS Wilayah untuk pengembangan.

Buat direktori baru dan beri nama `chatterbox`, lalu di dalamnya, buat direktori lain yang diberi nama `auth-server`.

Folder server kita akan memiliki struktur sebagai berikut:

```
- auth-server
  - src
    - main
      - kotlin
        - com
          - chatterbox
            - authserver
              - Application.kt
       - resources
         - application.conf
         - logback.xml
   - build.gradle.kts
```

Catatan: Anda dapat langsung menyalin/menempelkan kode di sini ke file yang direferensikan.

Selanjutnya, kita menambahkan semua dependensi dan plugin yang diperlukan agar server autentikasi berfungsi:

Skrip Kotlin:

```
// ./auth-server/build.gradle.kts

plugins {
   application
   kotlin("jvm")
   kotlin("plugin.serialization").version("1.7.10")
}

application {
   mainClass.set("io.ktor.server.netty.EngineMain")
}

dependencies {
   implementation("software.amazon.awssdk:ivschat:2.18.1")
   implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8:1.7.20")

   implementation("io.ktor:ktor-server-core:2.1.3")
   implementation("io.ktor:ktor-server-netty:2.1.3")
   implementation("io.ktor:ktor-server-content-negotiation:2.1.3")
   implementation("io.ktor:ktor-serialization-kotlinx-json:2.1.3")

   implementation("ch.qos.logback:logback-classic:1.4.4")
}
```

Sekarang kita perlu menyiapkan fungsionalitas pembuatan log untuk server autentikasi. (Untuk informasi selengkapnya, lihat Konfigurasikan pembuat log.)

XML:

```
// ./auth-server/src/main/resources/logback.xml

<configuration>
   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <encoder>
         <pattern>%d{YYYY-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</
pattern>
```

```
      </encoder>
   </appender>
   <root level="trace">
      <appender-ref ref="STDOUT"/>
   </root>
   <logger name="org.eclipse.jetty" level="INFO"/>
   <logger name="io.netty" level="INFO"/>
</configuration>
```

Server Ktor memerlukan pengaturan konfigurasi, yang dimuat dari file `application.*` di direktori `resources` secara otomatis, sehingga kita menambahkannya juga. (Untuk informasi selengkapnya, lihat Konfigurasi dalam file.)

HOCON:

```
// ./auth-server/src/main/resources/application.conf

ktor {
   deployment {
      port = 3000
   }
   application {
      modules = [ com.chatterbox.authserver.ApplicationKt.main ]
   }
}
```

Terakhir, mari kita implementasikan server kita:

Kotlin:

```
// ./auth-server/src/main/kotlin/com/chatterbox/authserver/Application.kt

package com.chatterbox.authserver

import io.ktor.http.*
import io.ktor.serialization.kotlinx.json.*
import io.ktor.server.application.*
import io.ktor.server.plugins.contentnegotiation.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import kotlinx.serialization.Serializable
import kotlinx.serialization.json.Json
```

```
import software.amazon.awssdk.services.ivschat.IvschatClient
import software.amazon.awssdk.services.ivschat.model.CreateChatTokenRequest

@Serializable
data class ChatTokenParams(var userId: String, var roomIdentifier: String)

@Serializable
data class ChatToken(
   val token: String,
   val sessionExpirationTime: String,
   val tokenExpirationTime: String,
)

fun Application.main() {
   install(ContentNegotiation) {
      json(Json)
   }

   routing {
      post("/create_chat_token") {
         val callParameters = call.receive<ChatTokenParams>()
         val request =
 CreateChatTokenRequest.builder().roomIdentifier(callParameters.roomIdentifier)
            .userId(callParameters.userId).build()
         val token = IvschatClient.create()
            .createChatToken(request)

         call.respond(
            ChatToken(
                token.token(),
                token.sessionExpirationTime().toString(),
                token.tokenExpirationTime().toString()
            )
         )
      }
   }
}
```

Membuat Proyek Chatterbox

Untuk membuat proyek Android, instal dan buka Android Studio.

Ikuti langkah-langkah yang tercantum dalam Panduan Membuat Proyek Android yang resmi.

- Di Pilih proyek Anda, pilih templat proyek Aktivitas Kosong untuk aplikasi Chatterbox kami.

- Di Konfigurasikan proyek Anda, pilih nilai berikut untuk bidang konfigurasi:
  - Nama: Aplikasi Saya
  - Nama paket: com.chatterbox.myapp
  - Simpan lokasi: arahkan ke direktori `chatterbox` yang dibuat di langkah sebelumnya
  - Bahasa: Kotlin
  - APITingkat minimum: API 21: Android 5.0 (Lollipop)

Setelah menentukan semua parameter konfigurasi dengan benar, struktur file kita di dalam folder `chatterbox` akan terlihat seperti berikut:

```
- app
  - build.gradle
  ...
- gradle
- .gitignore
- build.gradle
- gradle.properties
- gradlew
- gradlew.bat
- local.properties
- settings.gradle
- auth-server
  - src
    - main
      - kotlin
        - com
          - chatterbox
            - authserver
              - Application.kt
       - resources
         - application.conf
         - logback.xml
   - build.gradle.kts
```

Sekarang kita memiliki proyek Android yang berfungsi, kita dapat menambahkan com.amazonaws: ivs-chat-messaging dan org.jetbrains.kotlinx: ke dependensi kita. kotlinx-coroutines-core

`build.gradle` (Untuk informasi selengkapnya tentang kit alat build Gradle, lihat Mengonfigurasi build Anda.)

Catatan: Di bagian atas setiap cuplikan kode, ada jalur ke file tempat Anda harus membuat perubahan dalam proyek. Jalur tersebut bersifat relatif terhadap root proyek.

Kotlin:

```
// ./app/build.gradle

plugins {
// ...
}

android {
// ...
}

dependencies {
    implementation 'com.amazonaws:ivs-chat-messaging:1.1.0'
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.4'

// ...
}
```

Setelah dependensi baru ditambahkan, jalankan Sinkronkan Proyek dengan File Gradle di Android Studio untuk menyinkronkan proyek dengan dependensi baru. (Untuk informasi selengkapnya, lihat Menambahkan dependensi build.)

Agar mudah menjalankan server autentikasi (yang dibuat di bagian sebelumnya) dari root proyek, kita memasukkannya sebagai modul baru di `settings.gradle`. (Untuk informasi selengkapnya, lihat Menyusun dan Membangun Komponen Perangkat Lunak dengan Gradle.)

Skrip Kotlin:

```
// ./settings.gradle

// ...

rootProject.name = "My App"
include ':app'
```

```
include ':auth-server'
```

Mulai sekarang, karena `auth-server` disertakan dalam proyek Android, Anda dapat menjalankan server autentikasi dengan perintah berikut dari root proyek:

Shell:

```
./gradlew :auth-server:run
```

Hubungkan dengan Ruang Obrolan dan Amati Pembaruan Koneksi

Untuk membuka koneksi ruang obrolan, kami menggunakan callback siklus hidup aktivitas onCreate (), yang diaktifkan saat aktivitas pertama kali dibuat. ChatRoom Konstruktor mengharuskan kami untuk menyediakan `region` dan `tokenProvider` membuat instance koneksi ruangan.

Catatan: Fungsi `fetchChatToken` dalam cuplikan di bawah ini akan diimplementasikan di bagian berikutnya.

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

// AWS region of the room that was created in Getting Started with Amazon IVS Chat
const val REGION = "us-west-2"

class MainActivity : AppCompatActivity() {
    private var room: ChatRoom? = null
    // ...

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)

      // Create room instance
      room = ChatRoom(REGION, ::fetchChatToken)
   }

// ...
```

```
}
```

Menampilkan dan bereaksi terhadap perubahan dalam koneksi ruang obrolan adalah bagian penting dari pembuatan aplikasi obrolan, seperti `chatterbox`. Sebelum kita dapat mulai berinteraksi dengan ruang tersebut, kita harus berlangganan peristiwa status koneksi ruang obrolan untuk mendapatkan pembaruan.

Di Chat SDK for coroutine, `ChatRoom`mengharapkan kami untuk menangani peristiwa siklus hidup kamar di Flow. Untuk saat ini, fungsi tersebut hanya akan mencatat pesan konfirmasi, ketika diinvokasi:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

const val TAG = "Chatterbox-MyApp"

class MainActivity : AppCompatActivity() {
// ...

    override fun onCreate(savedInstanceState: Bundle?) {
        // ...

        // Create room instance
        room = ChatRoom(REGION, ::fetchChatToken).apply {
            lifecycleScope.launch {
                stateChanges().collect { state ->
                    Log.d(TAG, "state change to $state")
                }
            }

            lifecycleScope.launch {
                receivedMessages().collect { message ->
                    Log.d(TAG, "messageReceived $message")
                }
            }

            lifecycleScope.launch {
                receivedEvents().collect { event ->
```

```
                    Log.d(TAG, "eventReceived $event")
                }
            }

            lifecycleScope.launch {
                deletedMessages().collect { event ->
                    Log.d(TAG, "messageDeleted $event")
                }
            }

            lifecycleScope.launch {
                disconnectedUsers().collect { event ->
                    Log.d(TAG, "userDisconnected $event")
                }
            }
        }
    }
}
```

Setelah ini, kita perlu menyediakan kemampuan untuk membaca status koneksi ruang. Kami akan menyimpannya di `MainActivity.kt` properti dan menginisialisasi ke DISCONNECTED status default untuk kamar (lihat `ChatRoom` `state` di SDKReferensi Android IVS Obrolan). Agar tetap dapat memperbarui status lokal, kita perlu mengimplementasikan fungsi state-updater; sebut saja `updateConnectionState`:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

class MainActivity : AppCompatActivity() {
   private var connectionState = ChatRoom.State.DISCONNECTED

// ...

   private fun updateConnectionState(state: ChatRoom.State) {
      connectionState = state

     when (state) {
          ChatRoom.State.CONNECTED -> {
              Log.d(TAG, "room connected")
```

```
            }
            ChatRoom.State.DISCONNECTED -> {
                Log.d(TAG, "room disconnected")
            }
            ChatRoom.State.CONNECTING -> {
                Log.d(TAG, "room connecting")
            }
        }
}
```

Selanjutnya, kita mengintegrasikan fungsi state-updater kita dengan properti.listener: ChatRoom

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

class MainActivity : AppCompatActivity() {
// ...

    override fun onCreate(savedInstanceState: Bundle?) {
        // ...

        // Create room instance
        room = ChatRoom(REGION, ::fetchChatToken).apply {
            lifecycleScope.launch {
                stateChanges().collect { state ->
                    Log.d(TAG, "state change to $state")
                    updateConnectionState(state)

                }
            }

      // ...

      }
   }
}
```

Sekarang kita memiliki kemampuan untuk menyimpan, mendengarkan, dan bereaksi terhadap pembaruan ChatRoomstatus, saatnya untuk menginisialisasi koneksi:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

class MainActivity : AppCompatActivity() {
// ...

   private fun connect() {
      try {
         room?.connect()
      } catch (ex: Exception) {
         Log.e(TAG, "Error while calling connect()", ex)
      }
   }

   // ...
}
```

Membangun Penyedia Token

Kini saatnya membuat fungsi yang bertanggung jawab untuk membuat dan mengelola token obrolan di aplikasi. Dalam contoh ini kami menggunakan HTTPklien Retrofit untuk Android.

Sebelum kita dapat mengirim lalu lintas jaringan yang ada, kita harus menyiapkan konfigurasi keamanan jaringan untuk Android. (Untuk informasi selengkapnya, lihat Konfigurasikan keamanan jaringan.) Kita mulai dengan menambahkan izin jaringan ke file Manifes Aplikasi. Perhatikan tanda `user-permission` dan atribut `networkSecurityConfig` yang ditambahkan, yang akan mengarahkan ke konfigurasi keamanan jaringan baru kita. Dalam kode di bawah ini, ganti `<version>` dengan nomor versi Android Obrolan saat ini SDK (mis., 1.1.0).

XML:

```
// ./app/src/main/AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.chatterbox.myapp">
    <uses-permission android:name="android.permission.INTERNET" />
```

```
    <application
        android:allowBackup="true"
        android:fullBackupContent="@xml/backup_rules"
        android:label="@string/app_name"
        android:networkSecurityConfig="@xml/network_security_config"
// ...

// ./app/build.gradle


dependencies {
   implementation("com.amazonaws:ivs-chat-messaging:<version>")
// ...

   implementation("com.squareup.retrofit2:retrofit:2.9.0")
   implementation("com.squareup.retrofit2:converter-gson:2.9.0")
}
```

Nyatakan alamat IP Anda, misalnya domain `10.0.2.2` dan `localhost`, sebagai tepercaya untuk mulai bertukar pesan dengan backend kami:

XML:

```
// ./app/src/main/res/xml/network_security_config.xml

<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">10.0.2.2</domain>
        <domain includeSubdomains="true">localhost</domain>
    </domain-config>
</network-security-config>
```

Selanjutnya, kita perlu menambahkan dependensi baru, bersama dengan penambahan konverter Gson untuk respons HTTP parsing. Dalam kode di bawah ini, ganti `<version>` dengan nomor versi Android Obrolan saat ini SDK (mis., 1.1.0).

Skrip Kotlin:

```
// ./app/build.gradle

dependencies {
```

```
    implementation("com.amazonaws:ivs-chat-messaging:<version>")
// ...

    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
}
```

Untuk mengambil token obrolan, kita perlu membuat POST HTTP permintaan dari `chatterbox` aplikasi kita. Kita menentukan permintaan dalam antarmuka Retrofit yang akan diimplementasikan. (Lihat dokumentasi Retrofit. Juga biasakan diri Anda dengan spesifikasi CreateChatTokentitik akhir.)

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/network/ApiService.kt

package com.chatterbox.myapp.network

import com.amazonaws.ivs.chat.messaging.ChatToken
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.POST

data class CreateTokenParams(var userId: String, var roomIdentifier: String)

interface ApiService {
    @POST("create_chat_token")
    fun createChatToken(@Body params: CreateTokenParams): Call<ChatToken>
}


// ./app/src/main/java/com/chatterbox/myapp/network/RetrofitFactory.kt

package com.chatterbox.myapp.network

import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object RetrofitFactory {
    private const val BASE_URL = "http://10.0.2.2:3000"

    fun makeRetrofitService(): ApiService {
        return Retrofit.Builder()
            .baseUrl(BASE_URL)
```

```
            .addConverterFactory(GsonConverterFactory.create())
            .build().create(ApiService::class.java)
    }
}
```

Dengan jaringan yang telah siap, kini saatnya menambahkan fungsi yang bertanggung jawab untuk membuat dan mengelola token obrolan kita. Kita menambahkannya ke `MainActivity.kt`, yang secara otomatis dibuat ketika proyek dibuat:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import androidx.lifecycle.lifecycleScope
import kotlinx.coroutines.launch
import com.amazonaws.ivs.chat.messaging.*
import com.amazonaws.ivs.chat.messaging.coroutines.*
import com.chatterbox.myapp.network.CreateTokenParams
import com.chatterbox.myapp.network.RetrofitFactory
import retrofit2.Call
import java.io.IOException
import retrofit2.Callback
import retrofit2.Response

// custom tag for logging purposes
const val TAG = "Chatterbox-MyApp"

// any ID to be associated with auth token
const val USER_ID = "test user id"
// ID of the room the app wants to access. Must be an ARN. See Amazon Resource
 Names(ARNs)
const val ROOM_ID = "arn:aws:..."
// AWS region of the room that was created in Getting Started with Amazon IVS Chat
const val REGION = "us-west-2"

class MainActivity : AppCompatActivity() {

    private val service = RetrofitFactory.makeRetrofitService()
```

```
    private var userId: String = USER_ID

// ...

    private fun fetchChatToken(callback: ChatTokenCallback) {
        val params = CreateTokenParams(userId, ROOM_ID)
        service.createChatToken(params).enqueue(object : Callback<ChatToken> {
            override fun onResponse(call: Call<ChatToken>, response: Response<ChatToken>)
 {
                val token = response.body()
                if (token == null) {
                    Log.e(TAG, "Received empty token response")
                    callback.onFailure(IOException("Empty token response"))
                    return
                }

                Log.d(TAG, "Received token response $token")
                callback.onSuccess(token)
            }

            override fun onFailure(call: Call<ChatToken>, throwable: Throwable) {
                Log.e(TAG, "Failed to fetch token", throwable)
                callback.onFailure(throwable)
            }
        })
    }
}
```

Langkah Berikutnya

Sekarang, setelah Anda membuat koneksi ruang obrolan, lanjutkan ke Bagian 2 dari tutorial Coroutine Kotlin, Pesan dan Peristiwa.

IVSPesan Klien ObrolanSDK: Tutorial Kotlin Coroutines Bagian 2: Pesan dan Acara

Bagian kedua (dan terakhir) dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Membuat UI untuk Mengirim Pesan”
   a. the section called “Tata Letak Utama UI”
   b. the section called “Sel Teks Abstraksi UI untuk Menampilkan Teks Secara Konsisten”

   c. the section called “Pesan Obrolan Kiri UI”
   d. the section called “Pesan Kanan UI”
   e. the section called “Nilai Warna Tambahan UI”
2. the section called “Menerapkan Ikatan Tampilan”
3. the section called “Mengelola Permintaan Pesan-Obrolan”
4. the section called “Langkah Terakhir”

Untuk SDK dokumentasi lengkap, mulailah dengan Pesan Klien IVS Obrolan Amazon SDK (di sini di Panduan Pengguna IVS Obrolan Amazon) dan Pesan Klien Obrolan: SDK untuk Referensi Android (aktif GitHub).

Prasyarat

Pastikan Anda telah menyelesaikan Bagian 1 dari tutorial ini, Ruang Obrolan.

Membuat UI untuk Mengirim Pesan

Setelah kita berhasil menginisialisasi koneksi ruang obrolan, kini saatnya mengirim pesan pertama kami. UI diperlukan untuk fitur ini. Kami akan menambahkan:

- Tombol `connect/disconnect`
- Input pesan dengan tombol `send`
- Daftar pesan dinamis. Untuk membangun ini, kami menggunakan Android Jetpack RecyclerView.

Tata Letak Utama UI

Lihat Tata Letak Android Jetpack di dokumentasi developer Android.

XML:

```
// ./app/src/main/res/layout/activity_main.xml


<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://
schemas.android.com/apk/res/android"
                                                     xmlns:app="http://
schemas.android.com/apk/res-auto"
```

```
                                                    xmlns:tools="http://
schemas.android.com/tools"

 android:layout_width="match_parent"

 android:layout_height="match_parent">

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
                  xmlns:app="http://schemas.android.com/apk/res-auto"
                  android:id="@+id/connect_view"
                  android:layout_width="match_parent"
                  android:layout_height="match_parent"
                  android:gravity="center"
                  android:orientation="vertical">

        <androidx.cardview.widget.CardView
                android:id="@+id/connect_button"
                android:layout_width="match_parent"
                android:layout_height="48dp"
                android:layout_gravity=""
                android:layout_marginStart="16dp"
                android:layout_marginTop="4dp"
                android:layout_marginEnd="16dp"
                android:clickable="true"
                android:elevation="16dp"
                android:focusable="true"
                android:foreground="?android:attr/selectableItemBackground"
                app:cardBackgroundColor="@color/purple_500"
                app:cardCornerRadius="10dp">

            <TextView
                    android:id="@+id/connect_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_alignParentEnd="true"
                    android:layout_gravity="center"
                    android:layout_weight="1"
                    android:paddingHorizontal="12dp"
                    android:text="Connect"
                    android:textColor="@color/white"
                    android:textSize="16sp"/>

            <ProgressBar
                    android:id="@+id/activity_indicator"
```

```
                android:layout_width="20dp"
                android:layout_height="20dp"
                android:layout_gravity="center"
                android:layout_marginHorizontal="20dp"
                android:indeterminateOnly="true"
                android:indeterminateTint="@color/white"
                android:indeterminateTintMode="src_atop"
                android:keepScreenOn="true"
                android:visibility="gone"/>
    </androidx.cardview.widget.CardView>

</LinearLayout>

<androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/chat_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:clipToPadding="false"
        android:visibility="visible"
        tools:context=".MainActivity">

    <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            app:layout_constraintBottom_toTopOf="@+id/layout_message_input"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent">

        <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/recycler_view"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:clipToPadding="false"
                android:paddingTop="70dp"
                android:paddingBottom="20dp"/>
    </RelativeLayout>

    <RelativeLayout
            android:id="@+id/layout_message_input"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@android:color/white"
            android:clipToPadding="false"
```

```
                android:drawableTop="@android:color/black"
                android:elevation="18dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent">

            <EditText
                    android:id="@+id/message_edit_text"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_centerVertical="true"
                    android:layout_marginStart="16dp"
                    android:layout_toStartOf="@+id/send_button"
                    android:background="@android:color/transparent"
                    android:hint="Enter Message"
                    android:inputType="text"
                    android:maxLines="6"
                    tools:ignore="Autofill"/>

            <Button
                    android:id="@+id/send_button"
                    android:layout_width="84dp"
                    android:layout_height="48dp"
                    android:layout_alignParentEnd="true"
                    android:background="@color/black"
                    android:foreground="?android:attr/selectableItemBackground"
                    android:text="Send"
                    android:textColor="@color/white"
                    android:textSize="12dp"/>
        </RelativeLayout>
    </androidx.constraintlayout.widget.ConstraintLayout>


</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Sel Teks Abstraksi UI untuk Menampilkan Teks Secara Konsisten

XML:

```
// ./app/src/main/res/layout/common_cell.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
            android:id="@+id/layout_container"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@color/light_gray"
            android:minWidth="100dp"
            android:orientation="vertical">

    <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal">

        <TextView
                android:id="@+id/card_message_me_text_view"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_marginBottom="8dp"
                android:maxWidth="260dp"
                android:paddingLeft="12dp"
                android:paddingTop="8dp"
                android:paddingRight="12dp"
                android:text="This is a Message"
                android:textColor="#ffffff"
                android:textSize="16sp"/>

        <TextView
                android:id="@+id/failed_mark"
                android:layout_width="40dp"
                android:layout_height="match_parent"
                android:paddingRight="5dp"
                android:src="@drawable/ic_launcher_background"
                android:text="!"
                android:textAlignment="viewEnd"
                android:textColor="@color/white"
                android:textSize="25dp"
                android:visibility="gone"/>
    </LinearLayout>

</LinearLayout>
```

Pesan Obrolan Kiri UI

XML:

```
// ./app/src/main/res/layout/card_view_left.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:app="http://schemas.android.com/apk/res-auto"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_marginStart="8dp"
              android:layout_marginBottom="12dp"
              android:orientation="vertical">

    <TextView
            android:id="@+id/username_edit_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="UserName"/>

    <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

        <androidx.cardview.widget.CardView
                android:id="@+id/card_message_other"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="left"
                android:layout_marginBottom="4dp"
                android:foreground="?android:attr/selectableItemBackground"
                app:cardBackgroundColor="@color/light_gray_2"
                app:cardCornerRadius="10dp"
                app:cardElevation="0dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent">

            <include layout="@layout/common_cell"/>
        </androidx.cardview.widget.CardView>

        <TextView
                android:id="@+id/dateText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="4dp"
                android:layout_marginBottom="4dp"
```

```
                android:text="10:00"
                app:layout_constraintBottom_toBottomOf="@+id/card_message_other"
                app:layout_constraintLeft_toRightOf="@+id/card_message_other"/>
    </androidx.constraintlayout.widget.ConstraintLayout>


</LinearLayout>
```

Pesan Kanan UI

XML:

```
// ./app/src/main/res/layout/card_view_right.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://
schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
                android:layout_width="match_parent"
               android:layout_height="wrap_content"
android:layout_marginEnd="8dp">

    <androidx.cardview.widget.CardView
            android:id="@+id/card_message_me"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:layout_marginBottom="10dp"
            android:foreground="?android:attr/selectableItemBackground"
            app:cardBackgroundColor="@color/purple_500"
            app:cardCornerRadius="10dp"
            app:cardElevation="0dp"
            app:cardPreventCornerOverlap="false"
            app:cardUseCompatPadding="true"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent">

        <include layout="@layout/common_cell"/>

    </androidx.cardview.widget.CardView>

    <TextView
```

```
            android:id="@+id/dateText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginRight="12dp"
            android:layout_marginBottom="4dp"
            android:text="10:00"
            app:layout_constraintBottom_toBottomOf="@+id/card_message_me"
            app:layout_constraintRight_toLeftOf="@+id/card_message_me"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Nilai Warna Tambahan UI

XML:

```
// ./app/src/main/res/values/colors.xml

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--    ...-->
    <color name="dark_gray">#4F4F4F</color>
    <color name="blue">#186ED3</color>
    <color name="dark_red">#b30000</color>
    <color name="light_gray">#B7B7B7</color>
    <color name="light_gray_2">#eef1f6</color>
</resources>
```

Menerapkan Ikatan Tampilan

Kami memanfaatkan fitur Android View Binding untuk dapat mereferensikan class binding untuk XML tata letak kami. Untuk mengaktifkan fitur tersebut, atur opsi build `viewBinding` ke `true` pada `./app/build.gradle`:

Skrip Kotlin:

```
 // ./app/build.gradle

android {
//    ...

    buildFeatures {
```

```
        viewBinding = true
    }
//    ...
}
```

Sekarang saatnya menghubungkan UI dengan kode Kotlin:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
// ...

class MainActivity : AppCompatActivity() {
    // ...
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Create room instance
        room = ChatRoom(REGION, ::fetchChatToken).apply {
            // ...
        }

        binding.sendButton.setOnClickListener(::sendButtonClick)
        binding.connectButton.setOnClickListener {connect()}

        setUpChatView()

        updateConnectionState(ChatRoom.State.DISCONNECTED)
    }

    private fun sendMessage(request: SendMessageRequest) {
        lifecycleScope.launch {
           try {
               binding.messageEditText.text.clear()
               room?.awaitSendMessage(request)
           } catch (exception: ChatException) {
               Log.e(TAG, "Message rejected: ${exception.message}")
           } catch (exception: Exception) {
```

```
                Log.e(TAG, exception.message ?: "Unknown error occurred")
            }
        }
    }

    private fun sendButtonClick(view: View) {
        val content = binding.messageEditText.text.toString()
        if (content.trim().isEmpty()) {
            return
        }

        val request = SendMessageRequest(content)
        sendMessage(request)
    }
// ...

}
```

Kita juga menambahkan metode untuk menghapus pesan dan memutus koneksi pengguna dari obrolan, yang dapat diinvokasi menggunakan menu konteks pesan obrolan:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
//    ...

class MainActivity : AppCompatActivity() {
//    ...

    private fun deleteMessage(request: DeleteMessageRequest) {
        lifecycleScope.launch {
            try {
                room?.awaitDeleteMessage(request)
            } catch (exception: ChatException) {
                Log.e(TAG, "Delete message rejected: ${exception.message}")
            } catch (exception: Exception) {
                Log.e(TAG, exception.message ?: "Unknown error occurred")
            }
        }
    }
```

```
    private fun disconnectUser(request: DisconnectUserRequest) {
        lifecycleScope.launch {
           try {
               room?.awaitDisconnectUser(request)
           } catch (exception: ChatException) {
               Log.e(TAG, "Disconnect user rejected: ${exception.message}")
           } catch (exception: Exception) {
               Log.e(TAG, exception.message ?: "Unknown error occurred")
           }
        }
    }
}
```

Mengelola Permintaan Pesan-Obrolan

Kami membutuhkan cara untuk mengelola permintaan pesan-obrolan melalui semua kemungkinan statusnya:

- Tertunda — Pesan telah dikirim ke ruang obrolan, tetapi belum dikonfirmasi atau ditolak.
- Dikonfirmasi — Pesan dikirim oleh ruang obrolan ke semua pengguna (termasuk kita).
- Ditolak — Pesan ditolak oleh ruang obrolan dengan objek kesalahan.

Kita akan menyimpan permintaan obrolan dan pesan obrolan yang belum terselesaikan dalam suatu daftar. Daftar ini layak mendapat kelas terpisah, yang kita sebut `ChatEntries.kt`:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/ChatEntries.kt

package com.chatterbox.myapp

import com.amazonaws.ivs.chat.messaging.entities.ChatMessage
import com.amazonaws.ivs.chat.messaging.requests.SendMessageRequest

sealed class ChatEntry() {
    class Message(val message: ChatMessage) : ChatEntry()
    class PendingRequest(val request: SendMessageRequest) : ChatEntry()
    class FailedRequest(val request: SendMessageRequest) : ChatEntry()
}

class ChatEntries {
```

```
   /* This list is kept in sorted order. ChatMessages are sorted by date, while
pending and failed requests are kept in their original insertion point. */
   val entries = mutableListOf<ChatEntry>()
   var adapter: ChatListAdapter? = null

   val size get() = entries.size

   /**
    * Insert pending request at the end.
    */
   fun addPendingRequest(request: SendMessageRequest) {
       val insertIndex = entries.size
       entries.add(insertIndex, ChatEntry.PendingRequest(request))
       adapter?.notifyItemInserted(insertIndex)
   }

   /**
    * Insert received message at proper place based on sendTime. This can cause
removal of pending requests.
    */
   fun addReceivedMessage(message: ChatMessage) {
       /* Skip if we have already handled that message. */
       val existingIndex = entries.indexOfLast { it is ChatEntry.Message &&
it.message.id == message.id }
       if (existingIndex != -1) {
           return
       }

       val removeIndex = entries.indexOfLast {
           it is ChatEntry.PendingRequest && it.request.requestId == message.requestId
       }
       if (removeIndex != -1) {
           entries.removeAt(removeIndex)
       }

       val insertIndexRaw = entries.indexOfFirst { it is ChatEntry.Message &&
it.message.sendTime > message.sendTime }
       val insertIndex = if (insertIndexRaw == -1) entries.size else insertIndexRaw
       entries.add(insertIndex, ChatEntry.Message(message))

       if (removeIndex == -1) {
           adapter?.notifyItemInserted(insertIndex)
       } else if (removeIndex == insertIndex) {
           adapter?.notifyItemChanged(insertIndex)
```

```
        } else {
            adapter?.notifyItemRemoved(removeIndex)
            adapter?.notifyItemInserted(insertIndex)
        }
    }

    fun addFailedRequest(request: SendMessageRequest) {
        val removeIndex = entries.indexOfLast {
            it is ChatEntry.PendingRequest && it.request.requestId == request.requestId
        }
        if (removeIndex != -1) {
            entries.removeAt(removeIndex)
            entries.add(removeIndex, ChatEntry.FailedRequest(request))
            adapter?.notifyItemChanged(removeIndex)
        } else {
            val insertIndex = entries.size
            entries.add(insertIndex, ChatEntry.FailedRequest(request))
            adapter?.notifyItemInserted(insertIndex)
        }
    }

    fun removeMessage(messageId: String) {
        val removeIndex = entries.indexOfFirst { it is ChatEntry.Message &&
 it.message.id == messageId }
        entries.removeAt(removeIndex)
        adapter?.notifyItemRemoved(removeIndex)
    }

    fun removeFailedRequest(requestId: String) {
        val removeIndex = entries.indexOfFirst { it is ChatEntry.FailedRequest &&
 it.request.requestId == requestId }
        entries.removeAt(removeIndex)
        adapter?.notifyItemRemoved(removeIndex)
    }

    fun removeAll() {
        entries.clear()
    }
}
```

Untuk menghubungkan daftar dengan UI, kita menggunakan Adaptor. Untuk informasi selengkapnya, lihat Mengikat Data dengan AdapterView dan kelas pengikatan yang dihasilkan.

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/ChatListAdapter.kt

package com.chatterbox.myapp

import android.content.Context
import android.graphics.Color
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.LinearLayout
import android.widget.TextView
import androidx.core.content.ContextCompat
import androidx.core.view.isGone
import androidx.recyclerview.widget.RecyclerView
import com.amazonaws.ivs.chat.messaging.requests.DisconnectUserRequest
import java.text.DateFormat


class ChatListAdapter(
    private val entries: ChatEntries,
    private val onDisconnectUser: (request: DisconnectUserRequest) -> Unit,
) :
    RecyclerView.Adapter<ChatListAdapter.ViewHolder>() {
    var context: Context? = null
    var userId: String? = null

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val container: LinearLayout = view.findViewById(R.id.layout_container)
        val textView: TextView = view.findViewById(R.id.card_message_me_text_view)
        val failedMark: TextView = view.findViewById(R.id.failed_mark)
        val userNameText: TextView? = view.findViewById(R.id.username_edit_text)
        val dateText: TextView? = view.findViewById(R.id.dateText)
    }

    override fun onCreateViewHolder(viewGroup: ViewGroup, viewType: Int): ViewHolder {
        if (viewType == 0) {
            val rightView =
 LayoutInflater.from(viewGroup.context).inflate(R.layout.card_view_right, viewGroup,
 false)
            return ViewHolder(rightView)
        }
```

```
        val leftView =
LayoutInflater.from(viewGroup.context).inflate(R.layout.card_view_left, viewGroup,
false)
        return ViewHolder(leftView)
    }

    override fun getItemViewType(position: Int): Int {
        // Int 0 indicates to my message while Int 1 to other message
        val chatMessage = entries.entries[position]
        return if (chatMessage is ChatEntry.Message &&
chatMessage.message.sender.userId != userId) 1 else 0
    }

    override fun onBindViewHolder(viewHolder: ViewHolder, position: Int) {
        return when (val entry = entries.entries[position]) {
            is ChatEntry.Message -> {
                viewHolder.textView.text = entry.message.content

                val bgColor = if (entry.message.sender.userId == userId) {
                    R.color.purple_500
                } else {
                    R.color.light_gray_2
                }

viewHolder.container.setBackgroundColor(ContextCompat.getColor(context!!, bgColor))

                if (entry.message.sender.userId != userId) {
                    viewHolder.textView.setTextColor(Color.parseColor("#000000"))
                }

                viewHolder.failedMark.isGone = true

                viewHolder.itemView.setOnCreateContextMenuListener { menu, _, _ ->
                    menu.add("Kick out").setOnMenuItemClickListener {
                        val request =
DisconnectUserRequest(entry.message.sender.userId, "Some reason")
                        onDisconnectUser(request)
                        true
                    }
                }

                viewHolder.userNameText?.text = entry.message.sender.userId
                viewHolder.dateText?.text =
```

```
 DateFormat.getTimeInstance(DateFormat.SHORT).format(entry.message.sendTime)
            }

            is ChatEntry.PendingRequest -> {

 viewHolder.container.setBackgroundColor(ContextCompat.getColor(context!!,
 R.color.light_gray))
                viewHolder.textView.text = entry.request.content
                viewHolder.failedMark.isGone = true
                viewHolder.itemView.setOnCreateContextMenuListener(null)
                viewHolder.dateText?.text = "Sending"
            }

            is ChatEntry.FailedRequest -> {
                viewHolder.textView.text = entry.request.content

 viewHolder.container.setBackgroundColor(ContextCompat.getColor(context!!,
 R.color.dark_red))
                viewHolder.failedMark.isGone = false
                viewHolder.dateText?.text = "Failed"
            }
        }
    }

    override fun onAttachedToRecyclerView(recyclerView: RecyclerView) {
        super.onAttachedToRecyclerView(recyclerView)
        context = recyclerView.context
    }

    override fun getItemCount() = entries.entries.size
}
```

Langkah Terakhir

Saatnya menghubungkan adaptor baru kita, dengan mengikat kelas `ChatEntries` ke `MainActivity`:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt

package com.chatterbox.myapp
```

```
// ...

import com.chatterbox.myapp.databinding.ActivityMainBinding
import com.chatterbox.myapp.ChatListAdapter
import com.chatterbox.myapp.ChatEntries

class MainActivity : AppCompatActivity() {
    // ...
    private var entries = ChatEntries()
    private lateinit var adapter: ChatListAdapter

    // ...

    private fun setUpChatView() {
        adapter = ChatListAdapter(entries, ::disconnectUser)
        entries.adapter = adapter

        val recyclerViewLayoutManager = LinearLayoutManager(this@MainActivity,
 LinearLayoutManager.VERTICAL, false)
        binding.recyclerView.layoutManager = recyclerViewLayoutManager
        binding.recyclerView.adapter = adapter

        binding.sendButton.setOnClickListener(::sendButtonClick)
        binding.messageEditText.setOnEditorActionListener { _, _, event ->
            val isEnterDown = (event.action == KeyEvent.ACTION_DOWN) && (event.keyCode
 == KeyEvent.KEYCODE_ENTER)
            if (!isEnterDown) {
                return@setOnEditorActionListener false
            }

            sendButtonClick(binding.sendButton)
            return@setOnEditorActionListener true
        }
    }
}
```

Karena kami sudah memiliki kelas yang bertanggung jawab untuk melacak permintaan obrolan kami (`ChatEntries`), kami siap menerapkan kode untuk memanipulasi`entries`. roomListener Kita akan memperbarui `entries` dan `connectionState` sesuai dengan peristiwa yang kita tanggapi:

Kotlin:

```
// ./app/src/main/java/com/chatterbox/myapp/MainActivity.kt
```

```
package com.chatterbox.myapp
// ...

class MainActivity : AppCompatActivity() {
// ...


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Create room instance
        room = ChatRoom(REGION, ::fetchChatToken).apply {
            lifecycleScope.launch {
                stateChanges().collect { state ->
                    Log.d(TAG, "state change to $state")
                    updateConnectionState(state)
                    if (state == ChatRoom.State.DISCONNECTED) {
                       entries.removeAll()
                    }
                }
            }

            lifecycleScope.launch {
                receivedMessages().collect { message ->
                    Log.d(TAG, "messageReceived $message")
                    entries.addReceivedMessage(message)
                }
            }

            lifecycleScope.launch {
                receivedEvents().collect { event ->
                    Log.d(TAG, "eventReceived $event")
                }
            }

            lifecycleScope.launch {
                deletedMessages().collect { event ->
                    Log.d(TAG, "messageDeleted $event")
                    entries.removeMessage(event.messageId)
                }
            }
```

```
            lifecycleScope.launch {
                disconnectedUsers().collect { event ->
                    Log.d(TAG, "userDisconnected $event")
                }
            }
        }

        binding.sendButton.setOnClickListener(::sendButtonClick)
        binding.connectButton.setOnClickListener {connect()}

        setUpChatView()

        updateConnectionState(ChatRoom.State.DISCONNECTED)
    }

// ...

}
```

Sekarang Anda seharusnya dapat menjalankan aplikasi! (Lihat Membangun dan menjalankan aplikasi Anda.) Ingatlah untuk menjalankan server backend Anda saat menggunakan aplikasi. Anda dapat memutarnya dari terminal di root proyek kita dengan perintah ini: `./gradlew :auth-server:run` atau dengan menjalankan tugas Gradle `auth-server:run` langsung dari Android Studio.

IVSPesan Klien ObrolanSDK: Panduan iOS

Amazon Interactive Video (IVS) Chat Client Messaging iOS SDK menyediakan antarmuka yang memungkinkan Anda menggabungkan Pesan IVS Obrolan kami API di platform menggunakan bahasa pemrograman Swift Apple.

Versi terbaru dari IVS Chat Client Messaging iOSSDK: 1.0.0 (Catatan Rilis)

Dokumentasi dan tutorial referensi: Untuk informasi tentang metode terpenting yang tersedia di iOS Pesan Klien IVS Obrolan AmazonSDK, lihat dokumentasi referensi di: https://aws.github.io/ amazon-ivs-chat-messaging -sdk-ios/1.0.0/. Repositori ini juga berisi berbagai artikel dan tutorial.

Contoh kode: Lihat contoh repositori iOS di GitHub: https://github.com/aws-samples/ amazon-ivs-chat-for -ios-demo.

Persyaratan platform: iOS 13.0 atau lebih tinggi diperlukan untuk pengembangan.

Memulai dengan IVS Chat Client Messaging iOS SDK

Kami menyarankan Anda mengintegrasikan SDK via Swift Package Manager. Atau, Anda dapat menggunakan CocoaPodsatau mengintegrasikan kerangka kerja secara manual.

Setelah mengintegrasikanSDK, Anda dapat mengimpor SDK dengan menambahkan kode berikut di bagian atas file Swift Anda yang relevan:

```
import AmazonIVSChatMessaging
```

Manajer Paket Swift

Untuk menggunakan pustaka `AmazonIVSChatMessaging` dalam proyek Manajer Paket Swift, tambahkan pustaka tersebut ke dependensi untuk paket Anda dan dependensi untuk target yang relevan:

1. Unduh yang terbaru `.xcframework` dari https://ivschat.live-video.net/1.0.0/A mazonIVSChat Messaging.XCFramework.zip.
2. Di Terminal Anda, jalankan:

   ```
   shasum -a 256 path/to/downloaded/AmazonIVSChatMessaging.xcframework.zip
   ```

3. Ambil output dari langkah sebelumnya dan tempelkan ke properti checksum `.binaryTarget` seperti yang ditunjukkan di bawah ini dalam file `Package.swift` proyek Anda:

   ```
   let package = Package(
      // name, platforms, products, etc.
      dependencies: [
         // other dependencies
      ],
      targets: [
         .target(
            name: "<target-name>",
            dependencies: [
               // If you want to only bring in the SDK
               .binaryTarget(
                  name: "AmazonIVSChatMessaging",
                  url: "https://ivschat.live-video.net/1.0.0/
   AmazonIVSChatMessaging.xcframework.zip",
   ```

```
                checksum: "<SHA-extracted-using-steps-detailed-above>"
            ),
            // your other dependencies
        ],
    ),
    // other targets
]
)
```

CocoaPods

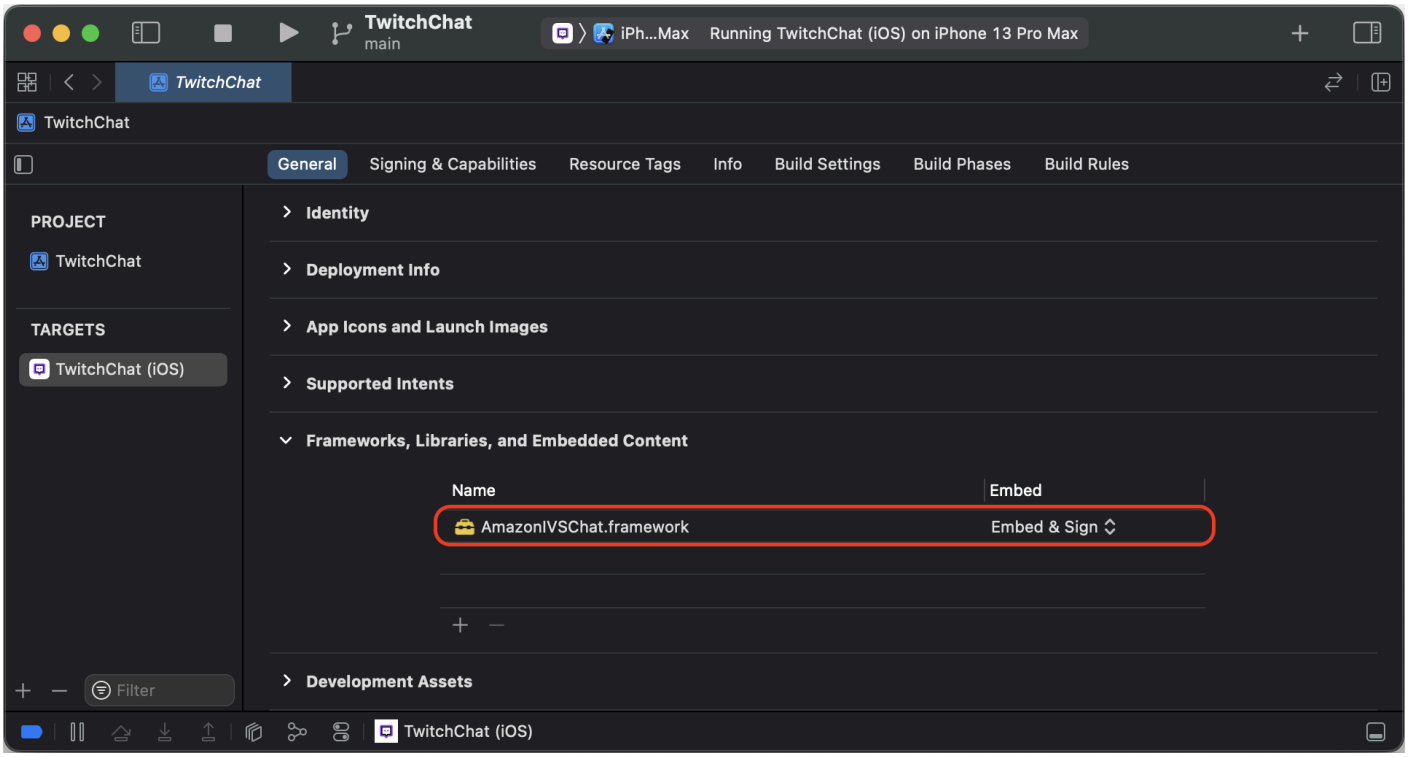
Rilis diterbitkan melalui CocoaPods di bawah nama`AmazonIVSChatMessaging`. Tambahkan dependensi ini ke Podfile Anda:

```
pod 'AmazonIVSChat'
```

Jalankan `pod install` dan SDK akan tersedia di Anda`.xcworkspace`.

Instalasi Manual

1. Unduh versi terbaru dari https://ivschat.live-video.net/1.0.0/A mazonIVSChat Messaging.XCFramework.zip.
2. Ekstrak isi arsip. `AmazonIVSChatMessaging.xcframework`berisi SDK untuk perangkat dan simulator.
3. Sematkan `AmazonIVSChatMessaging.xcframework` yang telah diekstrak dengan menyeretnya ke bagian Kerangka Kerja, Pustaka, dan Konten Tersemat pada tab Umum untuk target aplikasi Anda:



Menggunakan IVS Chat Client Messaging iOS SDK

Dokumen ini membawa Anda melalui langkah-langkah yang terlibat dalam menggunakan pesan klien IVS obrolan Amazon iOSSDK.

Menghubungkan ke Ruang Obrolan

Sebelum memulai, Anda harus terbiasa dengan Memulai dengan IVS Obrolan Amazon. Lihat juga contoh aplikasi untuk Web, Android, dan iOS.

Agar dapat terhubung ke ruang obrolan, aplikasi Anda memerlukan beberapa cara pengambilan token obrolan yang disediakan oleh backend. Aplikasi Anda mungkin akan mengambil token obrolan menggunakan permintaan jaringan ke backend Anda.

Untuk mengkomunikasikan token obrolan yang diambil ini denganSDK, `ChatRoom` model mengharuskan Anda untuk menyediakan `async` fungsi atau instance objek yang sesuai dengan `ChatTokenProvider` protokol yang disediakan pada titik inisialisasi. SDK Nilai yang dikembalikan oleh salah satu metode ini harus menjadi contoh dari SDK `ChatToken` model.

Catatan: Anda mengisi instans model `ChatToken` menggunakan data yang diambil dari backend. Bidang yang diperlukan untuk menginisialisasi `ChatToken` instance sama dengan bidang dalam

CreateChatTokenrespons. Untuk informasi selengkapnya tentang menginisialisasi instance `ChatToken` model, lihat Membuat instance dari. ChatToken Ingat, backend Anda bertanggung jawab untuk menyediakan data dalam respons `CreateChatToken` terhadap aplikasi. Cara Anda memutuskan untuk berkomunikasi dengan backend guna menghasilkan token obrolan bergantung pada aplikasi dan infrastrukturnya.

Setelah memilih strategi Anda untuk memberikanSDK, hubungi `.connect()` setelah berhasil menginisialisasi `ChatRoom` instance dengan penyedia token Anda dan AWSwilayah yang digunakan backend Anda untuk membuat ruang obrolan yang Anda coba sambungkan. `ChatToken` Perhatikan bahwa `.connect()` adalah fungsi asikron lemparan:

```
import AmazonIVSChatMessaging

let room = ChatRoom(
   awsRegion: <region-your-backend-created-the-chat-room-in>,
   tokenProvider: <your-chosen-token-provider-strategy>
)
try await room.connect()
```

Sesuai dengan Protokol ChatTokenProvider

Untuk parameter `tokenProvider` dalam penginisialisasi untuk `ChatRoom`, Anda dapat menyediakan instans `ChatTokenProvider`. Berikut adalah contoh dari objek yang sesuai dengan `ChatTokenProvider`:

```
import AmazonIVSChatMessaging

// This object should exist somewhere in your app
class ChatService: ChatTokenProvider {
   func getChatToken() async throws -> ChatToken {
      let request = YourApp.getTokenURLRequest
      let data = try await URLSession.shared.data(for: request).0
      ...
      return ChatToken(
         token: String(data: data, using: .utf8)!,
         tokenExpirationTime: ..., // this is optional
         sessionExpirationTime: ... // this is optional
      )
   }
}
```

Anda kemudian dapat mengambil instans dari objek yang sesuai ini dan memberikannya ke penginisialisasi untuk `ChatRoom`:

```
// This should be the same AWS Region that you used to create
// your Chat Room in the Control Plane
let awsRegion = "us-west-2"
let service = ChatService()
let room = ChatRoom(
   awsRegion: awsRegion,
   tokenProvider: service
)
try await room.connect()
```

Menyediakan Fungsi asinkron di Swift

Misalkan Anda sudah memiliki manajer yang digunakan untuk mengelola permintaan jaringan aplikasi Anda. Manajer tersebut mungkin akan terlihat seperti ini:

```
import AmazonIVSChatMessaging

class EndpointManager {
   func getAccounts() async -> AppUser {...}
   func signIn(user: AppUser) async {...}
   ...
}
```

Anda hanya dapat menambahkan fungsi lain di manajer untuk mengambil `ChatToken` dari backend:

```
import AmazonIVSChatMessaging

class EndpointManager {
   ...
   func retrieveChatToken() async -> ChatToken {...}
}
```

Kemudian, gunakan referensi untuk fungsi tersebut di Swift saat menginisialisasi `ChatRoom`:

```
import AmazonIVSChatMessaging

let endpointManager: EndpointManager
let room = ChatRoom(
```

```
    awsRegion: endpointManager.awsRegion,
    tokenProvider: endpointManager.retrieveChatToken
)
try await room.connect()
```

Buat sebuah Instance dari ChatToken

Anda dapat dengan mudah membuat instance `ChatToken` menggunakan penginisialisasi yang disediakan di. SDK Lihat dokumentasi di `Token.swift` untuk mempelajari selengkapnya tentang properti di `ChatToken`.

```
import AmazonIVSChatMessaging

let chatToken = ChatToken(
    token: <token-string-retrieved-from-your-backend>,
    tokenExpirationTime: nil, // this is optional
    sessionExpirationTime: nil // this is optional
)
```

Menggunakan Decodable

Jika, saat berinteraksi dengan IVS ObrolanAPI, backend Anda memutuskan untuk hanya meneruskan CreateChatTokenrespons ke aplikasi frontend Anda, Anda dapat memanfaatkan kesesuaian dengan protokol `ChatToken` Swift. `Decodable` Namun, ada hal yang harus diperhatikan.

Payload `CreateChatToken` respons menggunakan string untuk tanggal yang diformat menggunakan standar ISO8601 untuk stempel waktu internet. Di Swift, biasanya Anda perlu menyediakan `JSONDecoder.DateDecodingStrategy.iso8601` sebagai nilai untuk properti `JSONDecoder`, yaitu `.dateDecodingStrategy`. Namun, `CreateChatToken` menggunakan detik pecahan presisi tinggi dalam string-nya, dan ini tidak didukung oleh `JSONDecoder.DateDecodingStrategy.iso8601`.

Untuk kenyamanan Anda, SDK menyediakan ekstensi publik `JSONDecoder.DateDecodingStrategy` dengan `.preciseISO8601` strategi tambahan yang memungkinkan Anda untuk berhasil menggunakan `JSONDecoder` saat mendekode instance: `ChatToken`

```
import AmazonIVSChatMessaging
```

```
// The CreateChatToken data forwarded by your backend
let responseData: Data

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .preciseISO8601
let token = try decoder.decode(ChatToken.self, from: responseData)
```

Memutus Koneksi dari Ruang Obrolan

Untuk memutus koneksi secara manual dari instans `ChatRoom` yang berhasil Anda sambungkan, panggil `room.disconnect()`. Secara default, ruang obrolan secara otomatis memanggil fungsi ini saat tidak dialokasikan.

```
import AmazonIVSChatMessaging

let room = ChatRoom(...)
try await room.connect()

// Disconnect
room.disconnect()
```

Menerima Pesan/Peristiwa Obrolan

Untuk mengirim dan menerima pesan di ruang obrolan, Anda perlu menyediakan objek yang sesuai dengan protokol `ChatRoomDelegate`, setelah Anda berhasil menginisialisasi instans `ChatRoom` dan memanggil `room.connect()`. Berikut adalah contoh umum penggunaan `UIViewController`:

```
import AmazonIVSChatMessaging
import Foundation
import UIKit

class ViewController: UIViewController {
   let room: ChatRoom = ChatRoom(
      awsRegion: "us-west-2",
      tokenProvider: EndpointManager.shared
   )

   override func viewDidLoad() {
      super.viewDidLoad()
      Task { try await setUpChatRoom() }
   }
```

```
   private func setUpChatRoom() async throws {
      // Set the delegate to start getting notifications for room events
      room.delegate = self
      try await room.connect()
   }
}

extension ViewController: ChatRoomDelegate {
   func room(_ room: ChatRoom, didReceive message: ChatMessage) { ... }
   func room(_ room: ChatRoom, didReceive event: ChatEvent) { ... }
   func room(_ room: ChatRoom, didDelete message: DeletedMessageEvent) { ... }
}
```

Mendapatkan Notifikasi saat Koneksi Berubah

Seperti yang diperkirakan, Anda tidak dapat melakukan tindakan seperti mengirim pesan di suatu ruang sampai ruang terhubung sepenuhnya. Arsitektur SDK mencoba untuk mendorong koneksi ke thread ChatRoom pada latar belakang melalui asyncAPIs. Jika Anda ingin membangun sesuatu di UI Anda yang menonaktifkan sesuatu seperti tombol kirim pesan, SDK ini menyediakan dua strategi untuk mendapatkan pemberitahuan saat status koneksi ruang obrolan berubah, menggunakan atau. `Combine` `ChatRoomDelegate` Hal ini dijelaskan di bawah ini.

Penting: Status koneksi ruang obrolan juga dapat berubah karena beberapa hal seperti koneksi jaringan yang terputus. Pertimbangkan hal ini saat membangun aplikasi Anda.

Menggunakan Kombinasi

Setiap instans `ChatRoom` hadir dengan penerbit `Combine` sendiri dalam bentuk properti `state`:

```
import AmazonIVSChatMessaging
import Combine

var cancellables: Set<AnyCancellable> = []

let room = ChatRoom(...)
room.state.sink { state in
   switch state {
   case .connecting:
      let image = UIImage(named: "antenna.radiowaves.left.and.right")
      sendMessageButton.setImage(image, for: .normal)
      sendMessageButton.isEnabled = false
```

```
    case .connected:
        let image = UIImage(named: "paperplane.fill")
        sendMessageButton.setImage(image, for: .normal)
        sendMessageButton.isEnabled = true
    case .disconnected:
        let image = UIImage(named: "antenna.radiowaves.left.and.right.slash")
        sendMessageButton.setImage(image, for: .normal)
        sendMessageButton.isEnabled = false
    }
}.assign(to: &cancellables)

// Connect to `ChatRoom` on a background thread
Task(priority: .background) {
    try await room.connect()
}
```

Menggunakan ChatRoomDelegate

Atau, gunakan fungsi opsional `roomDidConnect(_:)`, `roomIsConnecting(_:)`, dan `roomDidDisconnect(_:)` dalam objek yang sesuai dengan `ChatRoomDelegate`. Berikut adalah contoh penggunaan `UIViewController`:

```
import AmazonIVSChatMessaging
import Foundation
import UIKit

class ViewController: UIViewController {
    let room: ChatRoom = ChatRoom(
        awsRegion: "us-west-2",
        tokenProvider: EndpointManager.shared
    )

    override func viewDidLoad() {
        super.viewDidLoad()
        Task { try await setUpChatRoom() }
    }

    private func setUpChatRoom() async throws {
        // Set the delegate to start getting notifications for room events
        room.delegate = self
        try await room.connect()
    }
}
```

```
extension ViewController: ChatRoomDelegate {
   func roomDidConnect(_ room: ChatRoom) {
      print("room is connected!")
   }
   func roomIsConnecting(_ room: ChatRoom) {
      print("room is currently connecting or fetching a token")
   }
   func roomDidDisconnect(_ room: ChatRoom) {
      print("room disconnected!")
   }
}
```

Melakukan Tindakan di Ruang Obrolan

Pengguna yang berbeda memiliki kemampuan berbeda pula untuk tindakan yang dapat mereka lakukan di ruang obrolan; misalnya, mengirim pesan, menghapus pesan, atau memutus koneksi pengguna. Untuk melakukan salah satu tindakan ini, panggil `perform(request:)` yang terhubung`ChatRoom`, meneruskan instance dari salah satu `ChatRequest` objek yang disediakan di`SDK`. Permintaan yang didukung ada di `Request.swift`.

Beberapa tindakan yang dilakukan di ruang obrolan mewajibkan pengguna yang terhubung untuk memiliki kemampuan khusus yang diberikan kepada mereka saat aplikasi backend Anda memanggil `CreateChatToken`. Secara desain, SDK tidak dapat membedakan kemampuan pengguna yang terhubung. Oleh karena itu, sementara Anda dapat mencoba melakukan tindakan moderator dalam instance yang terhubung`ChatRoom`, bidang kontrol pada API akhirnya memutuskan apakah tindakan itu akan berhasil.

Semua tindakan yang melalui `room.perform(request:)` akan menunggu hingga ruang menerima instans model yang diharapkan (tipe yang dikaitkan dengan objek permintaan itu sendiri) yang cocok dengan `requestId` dari model yang diterima dan objek permintaan. Jika ada masalah dengan permintaan tersebut, `ChatRoom` selalu melemparkan kesalahan dalam bentuk `ChatError`. Definisi dari `ChatError` ada di `Error.swift`.

Mengirim Pesan

Untuk mengirim pesan obrolan, gunakan instans `SendMessageRequest`:

```
import AmazonIVSChatMessaging

let room = ChatRoom(...)
```

```
try await room.connect()
try await room.perform(
   request: SendMessageRequest(
      content: "Release the Kraken!"
   )
)
```

Seperti yang disebutkan di atas, `room.perform(request:)` kembali setelah `ChatMessage` yang sesuai diterima oleh `ChatRoom`. Jika ada masalah dengan permintaan (seperti melebihi batas karakter pesan untuk suatu ruang), instans `ChatError` akan dilemparkan untuk meresponsnya. Anda kemudian dapat memunculkan informasi yang berguna ini di UI:

```
import AmazonIVSChatMessaging

do {
   let message = try await room.perform(
      request: SendMessageRequest(
         content: "Release the Kraken!"
      )
   )
   print(message.id)
} catch let error as ChatError {
   switch error.errorCode {
   case .invalidParameter:
      print("Exceeded the character limit!")
   case .tooManyRequests:
      print("Exceeded message request limit!")
   default:
      break
   }

   print(error.errorMessage)
}
```

Menambahkan Metadata ke Pesan

Saat mengirim pesan, Anda dapat menambahkan metadata yang akan dikaitkan dengan pesan tersebut. `SendMessageRequest` memiliki properti `attributes`, yang dapat Anda gunakan untuk menginisialisasi permintaan. Data yang Anda lampirkan di sana dilampirkan pada pesan ketika orang lain menerima pesan tersebut di dalam ruang.

Berikut adalah contoh melampirkan data emote ke pesan yang sedang dikirim:

```
import AmazonIVSChatMessaging

let room = ChatRoom(...)
try await room.connect()
try await room.perform(
   request: SendMessageRequest(
      content: "Release the Kraken!",
      attributes: [
         "messageReplyId" : "<other-message-id>",
         "attached-emotes" : "krakenCry,krakenPoggers,krakenCheer"
      ]
   )
)
```

Penggunaan `attributes` dalam `SendMessageRequest` dapat sangat berguna untuk membangun fitur yang kompleks dalam produk obrolan Anda. Misalnya, seseorang dapat membangun fungsionalitas threading dengan menggunakan kamus atribut `[String : String]` di `SendMessageRequest`!

Muatan `attributes` sangat fleksibel dan kuat. Gunakan muatan tersebut untuk mendapatkan informasi tentang pesan yang tidak akan bisa Anda dapatkan dengan cara lain. Menggunakan atribut jauh lebih mudah daripada, misalnya, mengurai string pesan untuk mendapatkan informasi tentang hal-hal seperti emote.

Menghapus Pesan

Menghapus pesan obrolan sama seperti mengirim pesan. Gunakan fungsi `room.perform(request:)` pada `ChatRoom` untuk melakukan ini dengan membuat instans `DeleteMessageRequest`.

Agar dapat mengakses instans pesan Obrolan yang diterima sebelumnya dengan mudah, berikan nilai `message.id` ke penginisialisasi `DeleteMessageRequest`.

Atau, sediakan string alasan untuk `DeleteMessageRequest` agar Anda dapat memunculkannya di UI.

```
import AmazonIVSChatMessaging

let room = ChatRoom(...)
try await room.connect()
try await room.perform(
```

```
   request: DeleteMessageRequest(
      id: "<other-message-id-to-delete>",
      reason: "Abusive chat is not allowed!"
   )
)
```

Karena ini adalah tindakan moderator, pengguna Anda mungkin tidak benar-benar memiliki kemampuan untuk menghapus pesan pengguna lain. Anda dapat menggunakan mekanisme fungsi Swift yang dapat dilemparkan untuk memunculkan pesan kesalahan di UI saat pengguna mencoba menghapus pesan tanpa kemampuan yang sesuai.

Ketika backend Anda memanggil `CreateChatToken` untuk pengguna, backend harus meneruskan `"DELETE_MESSAGE"` ke bidang `capabilities` guna mengaktifkan fungsionalitas tersebut untuk pengguna obrolan yang terhubung.

Berikut adalah contoh tangkapan kesalahan kemampuan yang dilemparkan saat mencoba menghapus pesan tanpa izin yang sesuai:

```
import AmazonIVSChatMessaging

do {
   // `deleteEvent` is the same type as the object that gets sent to
   // `ChatRoomDelegate`'s `room(_:didDeleteMessage:)` function
   let deleteEvent = try await room.perform(
      request: DeleteMessageRequest(
         id: "<other-message-id-to-delete>",
         reason: "Abusive chat is not allowed!"
      )
   )
   dataSource.messages[deleteEvent.messageID] = nil
   tableView.reloadData()
} catch let error as ChatError {
   switch error.errorCode {
   case .forbidden:
      print("You cannot delete another user's messages. You need to be a mod to do
 that!")
   default:
      break
   }

   print(error.errorMessage)
}
```

Memutus Koneksi Pengguna Lain

Gunakan `room.perform(request:)` untuk memutus koneksi pengguna lain dari ruang obrolan. Secara khusus, gunakan instans `DisconnectUserRequest`. Semua `ChatMessage` yang diterima oleh `ChatRoom` memiliki properti `sender`, yang berisi ID pengguna yang harus Anda inisialisasi dengan instans `DisconnectUserRequest` secara tepat. Atau, sediakan string alasan untuk meminta pemutusan koneksi.

```
import AmazonIVSChatMessaging

let room = ChatRoom(...)
try await room.connect()

let message: ChatMessage = dataSource.messages["<message-id>"]
let sender: ChatUser = message.sender
let userID: String = sender.userId
let reason: String = "You've been disconnected due to abusive behavior"

try await room.perform(
   request: DisconnectUserRequest(
      id: userID,
      reason: reason
   )
)
```

Karena ini adalah contoh tindakan moderator yang lain, Anda dapat mencoba untuk memutus koneksi pengguna lain, tetapi tidak akan dapat melakukannya kecuali memiliki kemampuan `DISCONNECT_USER`. Kemampuan akan diatur ketika aplikasi backend Anda memanggil `CreateChatToken` dan memasukkan string `"DISCONNECT_USER"` ke bidang `capabilities`.

Jika pengguna Anda tidak memiliki kemampuan untuk memutus koneksi pengguna lain, `room.perform(request:)` akan melemparkan instans `ChatError`, seperti permintaan lainnya. Anda dapat memeriksa properti `errorCode` kesalahan untuk menentukan apakah permintaan gagal karena tidak mempunyai hak istimewa moderator:

```
import AmazonIVSChatMessaging

do {
   let message: ChatMessage = dataSource.messages["<message-id>"]
   let sender: ChatUser = message.sender
   let userID: String = sender.userId
```

```
    let reason: String = "You've been disconnected due to abusive behavior"

    try await room.perform(
        request: DisconnectUserRequest(
            id: userID,
            reason: reason
        )
    )
} catch let error as ChatError {
    switch error.errorCode {
    case .forbidden:
        print("You cannot disconnect another user. You need to be a mod to do that!")
    default:
        break
    }

    print(error.errorMessage)
}
```

IVSPesan Klien ObrolanSDK: Tutorial iOS

Amazon Interactive Video (IVS) Chat Client Messaging iOS SDK menyediakan antarmuka untuk memungkinkan Anda menggabungkan Pesan IVS Obrolan kami API di platform menggunakan bahasa pemrograman Swift Apple.

Untuk SDK tutorial Obrolan iOS, lihat https://aws.github. io/amazon-ivs-chat-messaging-sdk-ios/latest/tutorials/table-dari konten/.

IVSPesan Klien ObrolanSDK: JavaScript Panduan

Amazon Interactive Video (IVS) Chat Client Messaging JavaScript SDK memungkinkan Anda untuk menggabungkan Pesan IVS Obrolan Amazon kami API di platform menggunakan browser Web.

Versi terbaru dari IVS Chat Client Messaging JavaScriptSDK: 1.0.2 (Release Notes)

Dokumentasi referensi: Untuk informasi tentang metode terpenting yang tersedia di Amazon IVS Chat Client Messaging JavaScript SDK, lihat dokumentasi referensi di: https://aws.github.io/ amazon-ivs-chat-messaging -sdk-js/1.0.2/

Contoh kode: Lihat contoh repositori aktif GitHub, untuk demo khusus Web menggunakan: -demo JavaScript SDK https://github.com/aws-samples/ amazon-ivs-chat-web

Memulai Pesan Klien IVS Obrolan JavaScript SDK

Sebelum memulai, Anda harus terbiasa dengan Memulai dengan IVS Obrolan Amazon.

Menambahkan Paket

Gunakan:

```
$ npm install --save amazon-ivs-chat-messaging
```

atau:

```
$ yarn add amazon-ivs-chat-messaging
```

Dukungan React Native

IVSChat Client Messaging JavaScript SDK memiliki `uuid` ketergantungan yang menggunakan `crypto.getRandomValues` metode ini. Karena metode ini tidak didukung di React Native, Anda perlu menginstal polyfill tambahan `react-native-get-random-value` dan mengimpornya di bagian atas file `index.js`:

```
import 'react-native-get-random-values';
import {AppRegistry} from 'react-native';
import App from './src/App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

Siapkan Backend Anda

Integrasi ini memerlukan titik akhir di server Anda yang berbicara dengan IVSObrolan API Amazon. Gunakan AWSperpustakaan resmi untuk akses ke Amazon IVS API dari server Anda. Pustaka ini dapat diakses dalam beberapa bahasa dari paket publik; misalnya, node.js, java, dan go.

Buat endpoint server yang berbicara ke endpoint IVS Obrolan API CreateChatTokenAmazon, untuk membuat token obrolan bagi pengguna obrolan.

Menggunakan Pesan Klien IVS Obrolan JavaScript SDK

Dokumen ini membawa Anda melalui langkah-langkah yang terlibat dalam menggunakan pesan klien IVS obrolan Amazon JavaScript SDK.

Menginisialisasi Instans Ruang Obrolan

Buat instans dari kelas `ChatRoom`. Ini membutuhkan passing `regionOrUrl` (AWSwilayah tempat ruang obrolan Anda di-host) dan `tokenProvider` (metode pengambilan token akan dibuat pada langkah berikutnya):

```
const room = new ChatRoom({
  regionOrUrl: 'us-west-2',
  tokenProvider: tokenProvider,
});
```

Fungsi Penyedia Token

Buat fungsi penyedia token asinkron yang mengambil token obrolan dari backend Anda:

```
type ChatTokenProvider = () => Promise<ChatToken>;
```

Fungsi tersebut seharusnya tidak menerima parameter dan mengembalikan Promise yang berisi objek token obrolan:

```
type ChatToken = {
  token: string;
  sessionExpirationTime?: Date;
  tokenExpirationTime?: Date;
}
```

Fungsi ini diperlukan untuk menginisialisasi ChatRoom objek. Di bawah ini, isi bidang `<token>` dan `<date-time>` dengan nilai yang diterima dari backend Anda:

```
// You will need to fetch a fresh token each time this method is called by
// the IVS Chat Messaging SDK, since each token is only accepted once.
function tokenProvider(): Promise<ChatToken> {
  // Call you backend to fetch chat token from IVS Chat endpoint:
  // e.g. const token = await appBackend.getChatToken()
  return {
    token: "<token>",
    sessionExpirationTime: new Date("<date-time>"),
    tokenExpirationTime: new Date("<date-time>")
  }
}
```

Ingatlah untuk meneruskan `tokenProvider` ke ChatRoom konstruktor. ChatRoom menyegarkan token saat koneksi terputus atau sesi kedaluwarsa. Jangan gunakan `tokenProvider` untuk menyimpan token di mana saja; ChatRoom menanganinya untuk Anda.

Menerima Peristiwa

Selanjutnya, berlangganan peristiwa ruang obrolan untuk menerima peristiwa siklus hidup, serta pesan dan peristiwa yang dikirimkan di ruang obrolan:

```
/**
* Called when room is establishing the initial connection or reestablishing
* connection after socket failure/token expiration/etc
*/
const unsubscribeOnConnecting = room.addListener('connecting', () => { });

/** Called when connection has been established. */
const unsubscribeOnConnected = room.addListener('connect', () => { });

/** Called when a room has been disconnected. */
const unsubscribeOnDisconnected = room.addListener('disconnect', () => { });

/** Called when a chat message has been received. */
const unsubscribeOnMessageReceived = room.addListener('message', (message) => {
 /* Example message:
  * {
  *   id: "5OPsDdX18qcJ",
  *   sender: { userId: "user1" },
  *   content: "hello world",
  *   sendTime: new Date("2022-10-11T12:46:41.723Z"),
  *   requestId: "d1b511d8-d5ed-4346-b43f-49197c6e61de"
  * }
  */
});

/** Called when a chat event has been received. */
const unsubscribeOnEventReceived = room.addListener('event', (event) => {
 /* Example event:
  * {
  *   id: "5OPsDdX18qcJ",
  *   eventName: "customEvent,
  *   sendTime: new Date("2022-10-11T12:46:41.723Z"),
  *   requestId: "d1b511d8-d5ed-4346-b43f-49197c6e61de",
  *   attributes: { "Custom Attribute": "Custom Attribute Value" }
```

```
  * }
  */
});

/** Called when `aws:DELETE_MESSAGE` system event has been received. */
const unsubscribeOnMessageDelete = room.addListener('messageDelete',
 (deleteMessageEvent) => {
 /* Example delete message event:
  * {
  *   id: "AYk6xKitV4On",
  *   messageId: "R1BLTDN84zEO",
  *   reason: "Spam",
  *   sendTime: new Date("2022-10-11T12:56:41.113Z"),
  *   requestId: "b379050a-2324-497b-9604-575cb5a9c5cd",
  *   attributes: { MessageID: "R1BLTDN84zEO", Reason: "Spam" }
  * }
  */
});

/** Called when `aws:DISCONNECT_USER` system event has been received. */
const unsubscribeOnUserDisconnect = room.addListener('userDisconnect',
 (disconnectUserEvent) => {
 /* Example event payload:
  * {
  *   id: "AYk6xKitV4On",
  *   userId": "R1BLTDN84zEO",
  *   reason": "Spam",
  *   sendTime": new Date("2022-10-11T12:56:41.113Z"),
  *   requestId": "b379050a-2324-497b-9604-575cb5a9c5cd",
  *   attributes": { UserId: "R1BLTDN84zEO", Reason: "Spam" }
  * }
  */
});
```

Hubungkan ke Ruang Obrolan

Langkah terakhir dari inisialisasi dasar adalah menghubungkan ke ruang obrolan dengan membuat WebSocket koneksi. Untuk melakukan hal tersebut, cukup panggil metode `connect()` dalam instans ruang:

```
room.connect();
```

SDKAkan mencoba membuat koneksi ke ruang obrolan yang dikodekan dalam token obrolan yang diterima dari server Anda.

Setelah Anda memanggil `connect()`, ruang akan beralih ke status `connecting` dan mengeluarkan peristiwa `connecting`. Ketika ruang berhasil terhubung, ruang tersebut beralih ke status `connected` dan memancarkan peristiwa `connect`.

Kegagalan koneksi mungkin terjadi karena masalah saat mengambil token atau saat menghubungkan ke WebSocket. Dalam hal ini, ruang mencoba untuk menyambung kembali secara otomatis hingga berapa kali yang ditunjukkan oleh parameter konstruktor `maxReconnectAttempts`. Selama upaya penyambungan kembali, ruang berada dalam status `connecting` dan tidak mengeluarkan peristiwa tambahan. Setelah upaya penyambungan kembali habis, ruang bertransisi ke status `disconnected` dan mengeluarkan peristiwa `disconnect` (dengan alasan pemutusan koneksi yang relevan). Dalam status `disconnected`, ruang tidak lagi mencoba terhubung; Anda harus memanggil `connect()` lagi untuk memicu proses koneksi.

Melakukan Tindakan di Ruang Obrolan

Amazon IVS Chat Messaging SDK menyediakan tindakan pengguna untuk mengirim pesan, menghapus pesan, dan memutuskan koneksi pengguna lain. Tindakan ini tersedia pada instans `ChatRoom`. Tindakan ini mengembalikan objek `Promise` yang memungkinkan Anda untuk menerima konfirmasi atau penolakan permintaan.

Mengirim Pesan

Untuk permintaan ini, Anda harus memiliki kapasitas `SEND_MESSAGE` yang dienkode dalam token obrolan Anda.

Untuk memicu permintaan kirim-pesan:

```
const request = new SendMessageRequest('Test Echo');
room.sendMessage(request);
```

Untuk mendapatkan konfirmasi atau penolakan permintaan, `await` janji yang dikembalikan atau gunakan metode `then()`:

```
try {
  const message = await room.sendMessage(request);
  // Message was successfully sent to chat room
} catch (error) {
  // Message request was rejected. Inspect the `error` parameter for details.
```

```
}
```

Menghapus Pesan

Untuk permintaan ini, Anda harus memiliki kapasitas `DELETE_MESSAGE` yang dienkode dalam token obrolan Anda.

Guna menghapus pesan untuk tujuan moderasi, panggil metode `deleteMessage()`:

```
const request = new DeleteMessageRequest(messageId, 'Reason for deletion');
room.deleteMessage(request);
```

Untuk mendapatkan konfirmasi atau penolakan permintaan, `await` janji yang dikembalikan atau gunakan metode `then()`:

```
try {
  const deleteMessageEvent = await room.deleteMessage(request);
  // Message was successfully deleted from chat room
} catch (error) {
  // Delete message request was rejected. Inspect the `error` parameter for details.
}
```

Memutus Koneksi Pengguna Lain

Untuk permintaan ini, Anda harus memiliki kapasitas `DISCONNECT_USER` yang dienkode dalam token obrolan Anda.

Guna memutus koneksi pengguna lain untuk tujuan moderasi, panggil metode `disconnectUser()`:

```
const request = new DisconnectUserRequest(userId, 'Reason for disconnecting user');
room.disconnectUser(request);
```

Untuk mendapatkan konfirmasi atau penolakan permintaan, `await` janji yang dikembalikan atau gunakan metode `then()`:

```
try {
  const disconnectUserEvent = await room.disconnectUser(request);
  // User was successfully disconnected from the chat room
} catch (error) {
  // Disconnect user request was rejected. Inspect the `error` parameter for details.
}
```

Memutus Koneksi dari Ruang Obrolan

Untuk menutup koneksi Anda ke ruang obrolan, panggil metode `disconnect()` pada instans `room`:

```
room.disconnect();
```

Memanggil metode ini menyebabkan ruangan menutup yang mendasarinya WebSocket secara tertib. Instans ruang beralih ke status `disconnected` dan mengeluarkan peristiwa pemutusan koneksi, dengan alasan `disconnect` diatur menjadi `"clientDisconnect"`.

IVSPesan Klien ObrolanSDK: JavaScript Tutorial Bagian 1: Ruang Obrolan

Ini adalah bagian pertama dari tutorial dua bagian. Anda akan mempelajari hal-hal penting bekerja dengan Pesan Klien IVS Obrolan Amazon JavaScript SDK dengan membangun aplikasi yang berfungsi penuh menggunakan JavaScript/. TypeScript Kami menyebut aplikasi itu Chatterbox.

Audiens yang dituju adalah pengembang berpengalaman yang baru mengenal Amazon IVS Chat MessagingSDK. Anda harus merasa nyaman dengan bahasa TypeScript pemrograman JavaScript/ dan perpustakaan React.

Untuk singkatnya, kita akan merujuk ke Pesan Klien IVS Obrolan Amazon JavaScript SDK sebagai JS SDK Obrolan.

Catatan: Dalam beberapa kasus, contoh kode untuk JavaScript dan TypeScript identik, sehingga digabungkan.

Bagian pertama dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Menyiapkan Server Autentikasi/Otorisasi Lokal”
2. the section called “Membuat Proyek Chatterbox”
3. the section called “Menghubungkan ke Ruang Obrolan”
4. the section called “Membangun Penyedia Token”
5. the section called “Mengamati Pembaruan Koneksi”
6. the section called “Membuat Komponen Tombol Kirim”
7. the section called “Buat Input Pesan”
8. the section called “Langkah Berikutnya”

Untuk SDK dokumentasi lengkap, mulailah dengan Pesan Klien IVS Obrolan Amazon SDK (di sini di Panduan Pengguna IVS Obrolan Amazon) dan Pesan Klien Obrolan: SDK untuk JavaScript Referensi (aktif GitHub).

Prasyarat

- Biasakan diri dengan JavaScript/TypeScript dan library React. Jika Anda tidak terbiasa dengan React, pelajari dasar-dasar dalam Tic-Tac-Toe Tutorial ini.
- Baca dan pahami Memulai dengan IVS Chat.
- Buat AWS IAM pengguna dengan CreateRoom kemampuan CreateChatToken dan yang ditentukan dalam IAM kebijakan yang ada. (Lihat Memulai dengan IVS Chat).
- Pastikan kunci rahasia/akses untuk pengguna ini disimpan dalam file AWS kredensial. Untuk petunjuk, lihat Panduan AWS CLI Pengguna (terutama Konfigurasi dan pengaturan file kredenal).
- Buat ruang obrolan dan simpanARN. Lihat Memulai dengan IVS Chat. (Jika Anda tidak menyimpanARN, Anda dapat mencarinya nanti dengan konsol atau ObrolanAPI.)
- Instal lingkungan Node.js 14+ dengan manajer paket NPM atau Yarn.

Menyiapkan Server Autentikasi/Otorisasi Lokal

Aplikasi backend Anda bertanggung jawab untuk membuat ruang obrolan dan menghasilkan token obrolan yang diperlukan untuk JS SDK Obrolan untuk mengautentikasi dan mengotorisasi klien Anda untuk ruang obrolan Anda. Anda harus menggunakan backend Anda sendiri karena Anda tidak dapat menyimpan AWS kunci dengan aman di aplikasi seluler; penyerang canggih dapat mengekstrak ini dan mendapatkan akses ke akun Anda. AWS

Lihat Membuat Token Obrolan di Memulai IVS Obrolan Amazon. Seperti yang ditunjukkan pada diagram alur di sana, aplikasi sisi server Anda bertanggung jawab untuk membuat token obrolan. Hal ini berarti aplikasi Anda harus menyediakan caranya sendiri untuk menghasilkan token obrolan dengan memintanya dari aplikasi sisi server Anda.

Di bagian ini, Anda akan mempelajari dasar-dasar membuat penyedia token di backend Anda. Kami menggunakan kerangka kerja ekspres untuk membuat server lokal langsung yang mengelola pembuatan token obrolan menggunakan AWS lingkungan lokal Anda.

Buat `npm` proyek kosong menggunakanNPM. Buat direktori untuk menyimpan aplikasi Anda, dan jadikan sebagai direktori kerja Anda:

```
$ mkdir backend & cd backend
```

Gunakan `npm init` untuk membuat file `package.json` pada aplikasi Anda:

```
$ npm init
```

Perintah ini meminta beberapa hal pada Anda, termasuk nama dan versi aplikasi Anda. Untuk saat ini, cukup tekan RETURNuntuk menerima default untuk sebagian besar dari mereka, dengan pengecualian berikut:

```
entry point: (index.js)
```

Tekan RETURNuntuk menerima nama file default yang disarankan `index.js` atau masukkan nama file utama apa pun yang Anda inginkan.

Sekarang instal dependensi yang diperlukan:

```
$ npm install express aws-sdk cors dotenv
```

`aws-sdk` memerlukan variabel lingkungan konfigurasi, yang memuat secara otomatis dari file bernama `.env` yang terletak di direktori root. Untuk mengonfigurasikannya, buat file baru bernama `.env` dan isi informasi konfigurasi yang hilang:

```
# .env

# The region to send service requests to.
AWS_REGION=us-west-2

# Access keys use an access key ID and secret access key
# that you use to sign programmatic requests to AWS.

# AWS access key ID.
AWS_ACCESS_KEY_ID=...

# AWS secret access key.
AWS_SECRET_ACCESS_KEY=...
```

Sekarang kita buat file titik masuk di direktori root dengan nama yang Anda masukkan di atas dalam perintah `npm init`. Dalam hal ini, kami menggunakan `index.js`, dan mengimpor semua paket yang diperlukan:

```
// index.js
import express from 'express';
import AWS from 'aws-sdk';
import 'dotenv/config';
import cors from 'cors';
```

Sekarang, buat instans baru `express`:

```
const app = express();
const port = 3000;

app.use(express.json());
app.use(cors({ origin: ['http://127.0.0.1:5173'] }));
```

Setelah itu Anda dapat membuat POST metode endpoint pertama Anda untuk penyedia token. Ambil parameter yang diperlukan dari isi permintaan (`roomId`, `userId`, `capabilities`, dan `sessionDurationInMinutes`):

```
app.post('/create_chat_token', (req, res) => {
  const { roomIdentifier, userId, capabilities, sessionDurationInMinutes } = req.body
 || {};
});
```

Tambahkan validasi bidang yang wajib diisi:

```
app.post('/create_chat_token', (req, res) => {
  const { roomIdentifier, userId, capabilities, sessionDurationInMinutes } = req.body
 || {};

  if (!roomIdentifier || !userId) {
    res.status(400).json({ error: 'Missing parameters: `roomIdentifier`, `userId`' });
    return;
  }
});
```

Setelah menyiapkan POST metode, kami mengintegrasikan `createChatToken` dengan `aws-sdk` fungsionalitas inti otentikasi/otorisasi:

```
app.post('/create_chat_token', (req, res) => {
  const { roomIdentifier, userId, capabilities, sessionDurationInMinutes } = req.body
 || {};
```

```
  if (!roomIdentifier || !userId || !capabilities) {
    res.status(400).json({ error: 'Missing parameters: `roomIdentifier`, `userId`,
 `capabilities`' });
    return;
  }

  ivsChat.createChatToken({ roomIdentifier, userId, capabilities,
 sessionDurationInMinutes }, (error, data) => {
    if (error) {
      console.log(error);
      res.status(500).send(error.code);
    } else if (data.token) {
      const { token, sessionExpirationTime, tokenExpirationTime } = data;
      console.log(`Retrieved Chat Token: ${JSON.stringify(data, null, 2)}`);

      res.json({ token, sessionExpirationTime, tokenExpirationTime });
    }
  });
});
```

Di akhir file, tambahkan pendengar port untuk aplikasi `express` Anda:

```
app.listen(port, () => {
  console.log(`Backend listening on port ${port}`);
});
```

Sekarang Anda dapat menjalankan server dengan perintah berikut dari root proyek:

```
$ node index.js
```

Tip: Server ini menerima URL permintaan di https://localhost:3000.

Membuat Proyek Chatterbox

Pertama, Anda buat proyek React yang disebut `chatterbox`. Jalankan perintah ini:

```
npx create-react-app chatterbox
```

Anda dapat mengintegrasikan Chat Client Messaging JS SDK melalui Node Package Manager atau Yarn Package Manager:

- Npm: `npm install amazon-ivs-chat-messaging`
- Yarn: `yarn add amazon-ivs-chat-messaging`

Menghubungkan ke Ruang Obrolan

Di sini Anda membuat `ChatRoom` dan menghubungkannya menggunakan metode asinkron. `ChatRoom`Kelas mengelola koneksi pengguna Anda ke Chat JSSDK. Agar berhasil terhubung ke ruang obrolan, Anda harus menyediakan instans `ChatToken` dalam aplikasi React Anda.

Arahkan ke file App yang dibuat dalam proyek `chatterbox` default dan hapus semuanya di antara dua tanda `<div>`. Tidak memerlukan kode yang telah diisi sebelumnya. Saat ini, App kami cukup kosong.

```
// App.jsx / App.tsx

import * as React from 'react';

export default function App() {
  return <div>Hello!</div>;
}
```

Buat instans `ChatRoom` baru dan teruskan ke status menggunakan hook `useState`. Ini membutuhkan passing `regionOrUrl` (AWSwilayah di mana ruang obrolan Anda di-host) dan `tokenProvider` (digunakan untuk aliran otentikasi/otorisasi backend yang dibuat pada langkah selanjutnya).

Penting: Anda harus menggunakan AWS wilayah yang sama dengan wilayah tempat Anda membuat ruangan di Memulai dengan IVS Obrolan Amazon. APIIni adalah layanan AWS regional. Untuk daftar wilayah yang didukung dan titik akhir HTTPS layanan IVS Obrolan Amazon, lihat halaman wilayah IVS Obrolan Amazon.

```
// App.jsx / App.tsx

import React, { useState } from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';

export default function App() {
  const [room] = useState(() =>
    new ChatRoom({
      regionOrUrl: process.env.REGION as string,
```

```
      tokenProvider: () => {},
    }),
  );

  return <div>Hello!</div>;
}
```

Membangun Penyedia Token

Sebagai langkah berikutnya, kita perlu membangun fungsi `tokenProvider` tanpa parameter yang diperlukan oleh konstruktor `ChatRoom`. Pertama, kita akan membuat `fetchChatToken` fungsi yang akan membuat POST permintaan ke aplikasi backend yang Anda atur. the section called "Menyiapkan Server Autentikasi/Otorisasi Lokal" Token obrolan berisi informasi yang diperlukan SDK agar berhasil membuat koneksi ruang obrolan. Obrolan API menggunakan token ini sebagai cara aman untuk memvalidasi identitas pengguna, kemampuan dalam ruang obrolan, dan durasi sesi.

Di navigator Proyek, buat JavaScript file TypeScript/baru bernama`fetchChatToken`. Bangun permintaan pengambilan ke aplikasi `backend` dan kembalikan objek `ChatToken` dari respons. Tambahkan properti isi permintaan yang diperlukan untuk membuat token obrolan. Gunakan aturan yang ditentukan untuk Amazon Resource Names (ARNs). Properti ini didokumentasikan di CreateChatToken titik akhir.

Catatan: Yang URL Anda gunakan di sini sama dengan server lokal Anda URL yang dibuat ketika Anda menjalankan aplikasi backend.

TypeScript

```
// fetchChatToken.ts

import { ChatToken } from 'amazon-ivs-chat-messaging';

type UserCapability = 'DELETE_MESSAGE' | 'DISCONNECT_USER' | 'SEND_MESSAGE';

export async function fetchChatToken(
  userId: string,
  capabilities: UserCapability[] = [],
  attributes?: Record<string, string>,
  sessionDurationInMinutes?: number,
): Promise<ChatToken> {
  const response = await fetch(`${process.env.BACKEND_BASE_URL}/create_chat_token`,
 {
```

```
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      userId,
      roomIdentifier: process.env.ROOM_ID,
      capabilities,
      sessionDurationInMinutes,
      attributes
    }),
  });

  const token = await response.json();

  return {
    ...token,
    sessionExpirationTime: new Date(token.sessionExpirationTime),
    tokenExpirationTime: new Date(token.tokenExpirationTime),
  };
}
```

JavaScript

```
// fetchChatToken.js

export async function fetchChatToken(
  userId,
  capabilities = [],
  attributes,
  sessionDurationInMinutes) {
  const response = await fetch(`${process.env.BACKEND_BASE_URL}/create_chat_token`,
 {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      userId,
      roomIdentifier: process.env.ROOM_ID,
      capabilities,
```

```
      sessionDurationInMinutes,
      attributes
    }),
  });

  const token = await response.json();

  return {
    ...token,
    sessionExpirationTime: new Date(token.sessionExpirationTime),
    tokenExpirationTime: new Date(token.tokenExpirationTime),
  };
}
```

Mengamati Pembaruan Koneksi

Bereaksi terhadap perubahan status koneksi ruang obrolan adalah bagian penting dalam membuat aplikasi obrolan. Mari kita mulai dengan berlangganan peristiwa yang relevan:

```
// App.jsx / App.tsx

import React, { useState, useEffect } from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';
import { fetchChatToken } from './fetchChatToken';

export default function App() {
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION as string,
        tokenProvider: () => fetchChatToken('Mike', ['SEND_MESSAGE']),
      }),
  );

  useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {});
    const unsubscribeOnConnected = room.addListener('connect', () => {});
    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {});

    return () => {
      // Clean up subscriptions.
      unsubscribeOnConnecting();
```

```
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, [room]);

  return <div>Hello!</div>;
}
```

Selanjutnya, kita perlu menyediakan kemampuan untuk membaca status koneksi. Kita menggunakan hook `useState` untuk membuat beberapa status lokal di `App` dan mengatur status koneksi di dalam setiap pendengar.

TypeScript

```
// App.tsx

import React, { useState, useEffect } from 'react';
import { ChatRoom, ConnectionState } from 'amazon-ivs-chat-messaging';
import { fetchChatToken } from './fetchChatToken';

export default function App() {
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION as string,
        tokenProvider: () => fetchChatToken('Mike', ['SEND_MESSAGE']),
      }),
  );
  const [connectionState, setConnectionState] =
 useState<ConnectionState>('disconnected');

  useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {
      setConnectionState('connecting');
    });

    const unsubscribeOnConnected = room.addListener('connect', () => {
      setConnectionState('connected');
    });

    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
      setConnectionState('disconnected');
    });
```

```
    return () => {
      unsubscribeOnConnecting();
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, [room]);

  return <div>Hello!</div>;
}
```

JavaScript

```
// App.jsx

import React, { useState, useEffect } from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';
import { fetchChatToken } from './fetchChatToken';

export default function App() {
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION,
        tokenProvider: () => fetchChatToken('Mike', ['SEND_MESSAGE']),
      }),
  );
  const [connectionState, setConnectionState] = useState('disconnected');

  useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {
      setConnectionState('connecting');
    });

    const unsubscribeOnConnected = room.addListener('connect', () => {
      setConnectionState('connected');
    });

    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
      setConnectionState('disconnected');
    });

    return () => {
```

```
      unsubscribeOnConnecting();
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, [room]);

  return <div>Hello!</div>;
}
```

Setelah berlangganan status koneksi, tampilkan status koneksi dan hubungkan ke ruang obrolan dengan menggunakan metode `room.connect` di dalam hook `useEffect`:

```
// App.jsx / App.tsx

// ...

useEffect(() => {
  const unsubscribeOnConnecting = room.addListener('connecting', () => {
    setConnectionState('connecting');
  });

  const unsubscribeOnConnected = room.addListener('connect', () => {
    setConnectionState('connected');
  });

  const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
    setConnectionState('disconnected');
  });

  room.connect();

  return () => {
    unsubscribeOnConnecting();
    unsubscribeOnConnected();
    unsubscribeOnDisconnected();
  };
}, [room]);

// ...

return (
  <div>
```

```
    <h4>Connection State: {connectionState}</h4>
  </div>
);

// ...
```

Anda telah berhasil mengimplementasikan koneksi ruang obrolan.

Membuat Komponen Tombol Kirim

Di bagian ini, Anda membuat tombol kirim yang memiliki desain berbeda untuk setiap status koneksi. Tombol kirim memfasilitasi pengiriman pesan di ruang obrolan. Tombol ini juga berfungsi sebagai indikator visual mengenai apakah/kapan pesan dapat dikirim; misalnya, saat koneksi terputus atau sesi obrolan yang kedaluwarsa.

Pertama, buat file baru di direktori `src` proyek Chatterbox Anda dan beri nama `SendButton`. Kemudian, buat komponen yang akan menampilkan tombol untuk aplikasi obrolan Anda. Ekspor `SendButton` Anda dan impor ke `App`. Di `<div></div>` yang kosong, tambahkan `<SendButton />`.

TypeScript

```
// SendButton.tsx

import React from 'react';

interface Props {
  onPress?: () => void;
  disabled?: boolean;
}

export const SendButton = ({ onPress, disabled }: Props) => {
  return (
    <button disabled={disabled} onClick={onPress}>
      Send
    </button>
  );
};

// App.tsx

import { SendButton } from './SendButton';
```

```
// ...

return (
  <div>
    <div>Connection State: {connectionState}</div>
    <SendButton />
  </div>
);
```

JavaScript

```
// SendButton.jsx

import React from 'react';

export const SendButton = ({ onPress, disabled }) => {
  return (
    <button disabled={disabled} onClick={onPress}>
      Send
    </button>
  );
};

// App.jsx

import { SendButton } from './SendButton';

// ...

return (
  <div>
    <div>Connection State: {connectionState}</div>
    <SendButton />
  </div>
);
```

Selanjutnya di App, tentukan fungsi bernama onMessageSend dan berikan fungsi tersebut ke properti SendButton onPress. Tentukan variabel lain bernama isSendDisabled (yang mencegah pengiriman pesan ketika ruang tidak terhubung) dan berikan variabel tersebut ke properti SendButton disabled.

```
// App.jsx / App.tsx

// ...

const onMessageSend = () => {};

const isSendDisabled = connectionState !== 'connected';

return (
  <div>
    <div>Connection State: {connectionState}</div>
    <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
  </div>
);

// ...
```

Buat Input Pesan

Bilah pesan Chatterbox adalah komponen yang akan berinteraksi dengan Anda untuk mengirim pesan ke ruang obrolan. Biasanya, bilah ini berisi input teks untuk menulis pesan dan tombol untuk mengirim pesan Anda.

Untuk membuat komponen `MessageInput`, pertama buat file baru di direktori `src` dan beri nama `MessageInput`. Kemudian, buat komponen input terkontrol yang akan menampilkan input untuk aplikasi obrolan Anda. Ekspor `MessageInput` Anda dan impor ke `App` (di atas `<SendButton />`).

Buat status baru bernama `messageToSend` menggunakan hook `useState`, dengan string kosong sebagai nilai default-nya. Di bagian utama aplikasi Anda, berikan `messageToSend` ke `value` dari `MessageInput` dan berikan `setMessageToSend` ke properti `onMessageChange`:

TypeScript

```
// MessageInput.tsx

import * as React from 'react';

interface Props {
  value?: string;
  onValueChange?: (value: string) => void;
}
```

```
export const MessageInput = ({ value, onValueChange }: Props) => {
  return (
    <input type="text" value={value} onChange={(e) => onValueChange?.
(e.target.value)} placeholder="Send a message" />
  );
};


// App.tsx

// ...

import { MessageInput } from './MessageInput';

// ...

export default function App() {
  const [messageToSend, setMessageToSend] = useState('');

// ...

return (
  <div>
    <h4>Connection State: {connectionState}</h4>
    <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
    <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
  </div>
);
```

JavaScript

```
// MessageInput.jsx

import * as React from 'react';

export const MessageInput = ({ value, onValueChange }) => {
  return (
    <input type="text" value={value} onChange={(e) => onValueChange?.
(e.target.value)} placeholder="Send a message" />
  );
};
```

```
// App.jsx

// ...

import { MessageInput } from './MessageInput';

// ...

export default function App() {
  const [messageToSend, setMessageToSend] = useState('');

// ...


return (
  <div>
    <h4>Connection State: {connectionState}</h4>
    <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
    <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
  </div>
);
```

Langkah Berikutnya

Sekarang setelah Anda selesai membangun bilah pesan untuk Chatterbox, lanjutkan ke Bagian 2 dari JavaScript tutorial ini, Pesan dan Acara.

IVSPesan Klien ObrolanSDK: JavaScript Tutorial Bagian 2: Pesan dan Acara

Bagian kedua (dan terakhir) dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Berlangganan Peristiwa Pesan Obrolan”
2. the section called “Menampilkan Pesan yang Diterima”
   a. the section called “Membuat Komponen Pesan”
   b. the section called “Mengenali Pesan yang Dikirim oleh Pengguna Saat Ini”
   c. the section called “Membuat Komponen Daftar Pesan”
   d. the section called “Me-render Daftar Pesan Obrolan”

3. the section called “Melakukan Tindakan di Ruang Obrolan”
   a. the section called “Mengirim Pesan”
   b. the section called “Menghapus Pesan”
4. the section called “Langkah Berikutnya”

Catatan: Dalam beberapa kasus, contoh kode untuk JavaScript dan TypeScript identik, sehingga digabungkan.

Untuk SDK dokumentasi lengkap, mulailah dengan Pesan Klien IVS Obrolan Amazon SDK (di sini di Panduan Pengguna IVS Obrolan Amazon) dan Pesan Klien Obrolan: SDK untuk JavaScript Referensi (aktif GitHub).

Prasyarat

Pastikan Anda telah menyelesaikan Bagian 1 dari tutorial ini, Ruang Obrolan.

Berlangganan Peristiwa Pesan Obrolan

Instans `ChatRoom` menggunakan peristiwa untuk berkomunikasi ketika peristiwa terjadi di ruang obrolan. Untuk mulai mengimplementasikan pengalaman obrolan, Anda harus menunjukkan kepada pengguna saat orang lain mengirim pesan di ruang yang terhubung dengan mereka.

Di sini, Anda berlangganan peristiwa pesan obrolan. Selanjutnya, kami akan menunjukkan cara memperbarui daftar pesan yang Anda buat, yang diperbarui dengan setiap pesan/peristiwa.

Di App Anda, di dalam hook `useEffect`, berlangganan semua peristiwa pesan:

```
// App.tsx / App.jsx

useEffect(() => {
  // ...
  const unsubscribeOnMessageReceived = room.addListener('message', (message) => {});

  return () => {
    // ...
    unsubscribeOnMessageReceived();
  };
}, []);
```

Menampilkan Pesan yang Diterima

Menerima pesan adalah bagian inti dari pengalaman obrolan. Menggunakan Chat JSSDK, Anda dapat mengatur kode Anda untuk dengan mudah menerima acara dari pengguna lain yang terhubung ke ruang obrolan.

Selanjutnya, kami akan menunjukkan cara melakukan tindakan di ruang obrolan yang memanfaatkan komponen yang Anda buat di sini.

Di App Anda, tentukan status bernama `messages` dengan tipe array `ChatMessage` yang bernama `messages`:

TypeScript

```
// App.tsx

// ...

import { ChatRoom, ChatMessage, ConnectionState } from 'amazon-ivs-chat-messaging';

export default function App() {
  const [messages, setMessages] = useState<ChatMessage[]>([]);

  //...
}
```

JavaScript

```
// App.jsx

// ...

export default function App() {
  const [messages, setMessages] = useState([]);

  //...
}
```

Selanjutnya, di fungsi pendengar `message`, tambahkan `message` ke array `messages`:

```
// App.jsx / App.tsx
```

```
// ...

const unsubscribeOnMessageReceived = room.addListener('message', (message) => {
  setMessages((msgs) => [...msgs, message]);
});

// ...
```

Di bawah ini, kita akan menjalankan langkah demi langkah untuk menampilkan pesan yang diterima:

1. the section called “Membuat Komponen Pesan”
2. the section called “Mengenali Pesan yang Dikirim oleh Pengguna Saat Ini”
3. the section called “Membuat Komponen Daftar Pesan”
4. the section called “Me-render Daftar Pesan Obrolan”

Membuat Komponen Pesan

Komponen `Message` bertanggung jawab untuk me-render konten pesan yang diterima oleh ruang obrolan Anda. Di bagian ini, Anda membuat komponen pesan untuk me-render pesan obrolan individu di App.

Buat file baru di direktori `src` dan beri nama `Message`. Berikan tipe `ChatMessage` untuk komponen ini, dan berikan string `content` dari properti `ChatMessage` untuk menampilkan teks pesan yang diterima dari pendengar pesan ruang obrolan. Di Navigator Proyek, buka `Message`.

TypeScript

```
// Message.tsx

import * as React from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';

type Props = {
  message: ChatMessage;
}

export const Message = ({ message }: Props) => {
  return (
    <div style={{ backgroundColor: 'silver', padding: 6, borderRadius: 10, margin:
 10 }}>
```

```
      <p>{message.content}</p>
    </div>
  );
};
```

JavaScript

```
// Message.jsx

import * as React from 'react';

export const Message = ({ message }) => {
  return (
    <div style={{ backgroundColor: 'silver', padding: 6, borderRadius: 10, margin:
 10 }}>
      <p>{message.content}</p>
    </div>
  );
};
```

Tip: Gunakan komponen ini untuk menyimpan properti berbeda yang ingin Anda render di baris pesan; misalnya, avatarURLs, nama pengguna, dan stempel waktu saat pesan dikirim.

Mengenali Pesan yang Dikirim oleh Pengguna Saat Ini

Untuk mengenali pesan yang dikirim oleh pengguna saat ini, kita mengubah kode dan membuat konteks React untuk menyimpan `userId` pengguna saat ini.

Buat file baru di direktori `src` dan beri nama `UserContext`:

TypeScript

```
// UserContext.tsx

import React, { ReactNode, useState, useContext, createContext } from 'react';

type UserContextType = {
  userId: string;
  setUserId: (userId: string) => void;
};

const UserContext = createContext<UserContextType | undefined>(undefined);
```

```
export const useUserContext = () => {
  const context = useContext(UserContext);

  if (context === undefined) {
    throw new Error('useUserContext must be within UserProvider');
  }

  return context;
};

type UserProviderType = {
  children: ReactNode;
}

export const UserProvider = ({ children }: UserProviderType) => {
  const [userId, setUserId] = useState('Mike');

  return <UserContext.Provider value={{ userId, setUserId }}>{children}</
UserContext.Provider>;
};
```

JavaScript

```
// UserContext.jsx

import React, { useState, useContext, createContext } from 'react';

const UserContext = createContext(undefined);

export const useUserContext = () => {
  const context = useContext(UserContext);

  if (context === undefined) {
    throw new Error('useUserContext must be within UserProvider');
  }

  return context;
};

export const UserProvider = ({ children }) => {
  const [userId, setUserId] = useState('Mike');
```

```
  return <UserContext.Provider value={{ userId, setUserId }}>{children}</
UserContext.Provider>;
};
```

Catatan: Di sini kita menggunakan hook `useState` untuk menyimpan nilai `userId`. Ke depannya, Anda dapat menggunakan `setUserId` untuk mengubah konteks pengguna atau untuk tujuan login.

Selanjutnya, ganti `userId` pada parameter pertama yang diberikan ke `tokenProvider`, dengan menggunakan konteks yang dibuat sebelumnya:

```
// App.jsx / App.tsx

// ...

import { useUserContext } from './UserContext';

// ...


export default function App() {
  const [messages, setMessages] = useState<ChatMessage[]>([]);
  const { userId } = useUserContext();
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION,
        tokenProvider: () => tokenProvider(userId, ['SEND_MESSAGE']),
      }),
  );

  // ...
}
```

Dalam komponen `Message` Anda, gunakan `UserContext` yang dibuat sebelumnya, nyatakan variabel `isMine`, cocokkan `userId` pengirim dengan `userId` dari konteks, dan terapkan gaya pesan yang berbeda untuk pengguna saat ini.

TypeScript

```
// Message.tsx
```

```
import * as React from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

type Props = {
  message: ChatMessage;
}

export const Message = ({ message }: Props) => {
  const { userId } = useUserContext();

  const isMine = message.sender.userId === userId;

  return (
    <div style={{ backgroundColor: isMine ? 'lightblue' : 'silver', padding: 6,
 borderRadius: 10, margin: 10 }}>
      <p>{message.content}</p>
    </div>
  );
};
```

JavaScript

```
// Message.jsx

import * as React from 'react';
import { useUserContext } from './UserContext';

export const Message = ({ message }) => {
  const { userId } = useUserContext();

  const isMine = message.sender.userId === userId;

  return (
    <div style={{ backgroundColor: isMine ? 'lightblue' : 'silver', padding: 6,
 borderRadius: 10, margin: 10 }}>
      <p>{message.content}</p>
    </div>
  );
};
```

Membuat Komponen Daftar Pesan

Komponen `MessageList` bertanggung jawab untuk menampilkan percakapan ruang obrolan dari waktu ke waktu. File `MessageList` ini adalah kontainer yang menyimpan semua pesan kita. `Message` adalah satu baris di `MessageList`.

Buat file baru di direktori `src` dan beri nama `MessageList`. Tentukan `Props` dengan `messages` tipe array `ChatMessage`. Di dalam isi, petakan properti `messages` dan teruskan `Props` ke komponen `Message` Anda.

TypeScript

```
// MessageList.tsx

import React from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { Message } from './Message';

interface Props {
  messages: ChatMessage[];
}

export const MessageList = ({ messages }: Props) => {
  return (
    <div>
      {messages.map((message) => (
        <Message key={message.id} message={message}/>
      ))}
    </div>
  );
};
```

JavaScript

```
// MessageList.jsx

import React from 'react';
import { Message } from './Message';

export const MessageList = ({ messages }) => {
  return (
    <div>
```

```
      {messages.map((message) => (
        <Message key={message.id} message={message} />
      ))}
    </div>
  );
};
```

Me-render Daftar Pesan Obrolan

Sekarang bawa `MessageList` baru ke komponen App utama Anda:

```
// App.jsx / App.tsx

import { MessageList } from './MessageList';
// ...

return (
  <div style={{ display: 'flex', flexDirection: 'column', padding: 10 }}>
    <h4>Connection State: {connectionState}</h4>
    <MessageList messages={messages} />
    <div style={{ flexDirection: 'row', display: 'flex', width: '100%',
 backgroundColor: 'red' }}>
      <MessageInput value={messageToSend} onValueChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
    </div>
  </div>
);

// ...
```

Semua potongan puzzle sekarang sudah siap sehingga App Anda dapat mulai me-render pesan yang diterima oleh ruang obrolan Anda. Lanjutkan langkah di bawah ini untuk melihat cara melakukan tindakan di ruang obrolan yang memanfaatkan komponen yang telah Anda buat.

Melakukan Tindakan di Ruang Obrolan

Mengirim pesan dan melakukan tindakan moderator dalam ruang obrolan adalah beberapa cara utama Anda dalam berinteraksi dengan ruang obrolan. Di sini, Anda akan belajar cara menggunakan berbagai objek `ChatRequest` untuk melakukan tindakan umum di Chatterbox, seperti mengirim pesan, menghapus pesan, dan memutus koneksi pengguna lain.

Semua tindakan di ruang obrolan mengikuti pola umum: untuk setiap tindakan yang Anda lakukan di ruang obrolan, ada objek permintaan yang sesuai. Untuk setiap permintaan, ada objek respons yang sesuai yang Anda terima setelah konfirmasi permintaan.

Selama pengguna Anda diberi izin yang benar saat Anda membuat token obrolan, mereka akan berhasil melakukan tindakan yang sesuai menggunakan objek permintaan untuk melihat permintaan apa yang dapat Anda lakukan di ruang obrolan.

Di bawah ini, kami menjelaskan cara mengirim pesan dan menghapus pesan.

Mengirim Pesan

Kelas `SendMessageRequest` memungkinkan pengiriman pesan di ruang obrolan. Di sini, Anda memodifikasi App untuk mengirim permintaan pesan menggunakan komponen yang Anda buat di Buat Input Pesan (di Bagian 1 tutorial ini).

Untuk memulai, tentukan properti boolean baru bernama `isSending` dengan hook `useState`. Gunakan properti baru ini untuk mengaktifkan status nonaktif `button` HTML elemen Anda, menggunakan konstanta. `isSendDisabled` Di handler peristiwa untuk `SendButton` Anda, kosongkan nilai untuk `messageToSend` dan atur `isSending` ke true.

Karena Anda akan melakukan API panggilan dari tombol ini, menambahkan *`isSending`* boolean membantu mencegah beberapa API panggilan terjadi pada saat yang sama, dengan menonaktifkan interaksi pengguna pada Anda *`SendButton`* hingga permintaan selesai.

```
// App.jsx / App.tsx

// ...

const [isSending, setIsSending] = useState(false);

// ...

const onMessageSend = () => {
  setIsSending(true);
  setMessageToSend('');
};

// ...

const isSendDisabled = connectionState !== 'connected' || isSending;
```

```
// ...
```

Siapkan permintaan dengan membuat instans `SendMessageRequest` baru, dengan meneruskan konten pesan ke konstruktor. Setelah mengatur status `isSending` dan `messageToSend`, panggil metode `sendMessage`, yang mengirimkan permintaan ke ruang obrolan. Terakhir, hapus bendera `isSending` saat menerima konfirmasi atau penolakan permintaan.

TypeScript

```
// App.tsx

// ...
import { ChatMessage, ChatRoom, ConnectionState, SendMessageRequest } from 'amazon-
ivs-chat-messaging'
// ...

const onMessageSend = async () => {
  const request = new SendMessageRequest(messageToSend);
  setIsSending(true);
  setMessageToSend('');

  try {
    const response = await room.sendMessage(request);
  } catch (e) {
    console.log(e);
    // handle the chat error here...
  } finally {
    setIsSending(false);
  }
};

// ...
```

JavaScript

```
// App.jsx

// ...
import { ChatRoom, SendMessageRequest } from 'amazon-ivs-chat-messaging'
// ...

const onMessageSend = async () => {
```

```
  const request = new SendMessageRequest(messageToSend);
  setIsSending(true);
  setMessageToSend('');

  try {
    const response = await room.sendMessage(request);
  } catch (e) {
    console.log(e);
    // handle the chat error here...
  } finally {
    setIsSending(false);
  }
};

// ...
```

Jalankan Chatterbox: coba mengirim pesan dengan menyusun pesan menggunakan `MessageInput` Anda dan mengetuk `SendButton` Anda. Anda akan melihat pesan terkirim Anda di-render dalam `MessageList` yang dibuat sebelumnya.

Menghapus Pesan

Untuk menghapus pesan dari ruang obrolan, Anda harus memiliki kemampuan yang tepat. Kemampuan diberikan selama inisialisasi token obrolan yang Anda gunakan saat mengautentikasi ke ruang obrolan. Untuk keperluan bagian ini, `ServerApp` dari Menyiapkan Server Autentikasi/Otorisasi Lokal (di Bagian 1 tutorial ini) memungkinkan Anda menentukan kemampuan moderator. Hal ini dilakukan di aplikasi Anda menggunakan objek `tokenProvider` yang Anda buat di Membangun Penyedia Token (juga di Bagian 1).

Di sini, Anda memodifikasi `Message` dengan menambahkan fungsi untuk menghapus pesan.

Pertama, buka `App.tsx` dan tambahkan kemampuan `DELETE_MESSAGE`. (`capabilities` adalah parameter kedua dari fungsi `tokenProvider` Anda.)

Catatan: Ini adalah cara Anda `ServerApp` menginformasikan IVS Obrolan APIs bahwa pengguna yang dikaitkan dengan token obrolan yang dihasilkan dapat menghapus pesan di ruang obrolan. Dalam situasi dunia nyata, Anda mungkin akan memiliki logika backend yang lebih kompleks untuk mengelola kemampuan pengguna di infrastruktur aplikasi server Anda.

TypeScript

```
// App.tsx

// ...

const [room] = useState( () =>
    new ChatRoom({
      regionOrUrl: process.env.REGION as string,
      tokenProvider: () => tokenProvider(userId, ['SEND_MESSAGE',
 'DELETE_MESSAGE']),
    }),
);

// ...
```

JavaScript

```
// App.jsx

// ...

const [room] = useState( () =>
  new ChatRoom({
    regionOrUrl: process.env.REGION,
    tokenProvider: () => tokenProvider(userId, ['SEND_MESSAGE', 'DELETE_MESSAGE']),
  }),
);

// ...
```

Pada langkah berikutnya, Anda memperbarui `Message` untuk menampilkan tombol hapus.

Buka `Message` dan tentukan status boolean baru yang bernama `isDeleting` menggunakan hook `useState` dengan nilai awal `false`. Dengan menggunakan status ini, perbarui konten `Button` Anda menjadi berbeda, tergantung status `isDeleting` saat ini. Nonaktifkan tombol Anda ketika `isDeleting` true; hal ini mencegah Anda dari mencoba untuk membuat dua permintaan penghapusan pesan di waktu yang bersamaan.

TypeScript

```
// Message.tsx

import React, { useState } from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

type Props = {
  message: ChatMessage;
}

export const Message = ({ message }: Props) => {
  const { userId } = useUserContext();
  const [isDeleting, setIsDeleting] = useState(false);

  const isMine = message.sender.userId === userId;

  return (
    <div style={{ backgroundColor: isMine ? 'lightblue' : 'silver', padding: 6,
 borderRadius: 10, margin: 10 }}>
      <p>{message.content}</p>
      <button disabled={isDeleting}>Delete</button>
    </div>
  );
};
```

JavaScript

```
// Message.jsx

import React from 'react';
import { useUserContext } from './UserContext';

export const Message = ({ message }) => {
  const { userId } = useUserContext();
  const [isDeleting, setIsDeleting] = useState(false);

  return (
    <div style={{ backgroundColor: isMine ? 'lightblue' : 'silver', padding: 6,
 borderRadius: 10, margin: 10 }}>
      <p>{message.content}</p>
      <button disabled={isDeleting}>Delete</button>
```

```
    </div>
  );
};
```

Tentukan fungsi baru bernama `onDelete`, yang menerima string sebagai salah satu parameternya dan mengembalikan `Promise`. Di dalam isi penutupan tindakan `Button` Anda, gunakan `setIsDeleting` untuk mengalihkan boolean `isDeleting` sebelum dan sesudah panggilan ke `onDelete`. Untuk parameter string, berikan ID pesan komponen Anda.

TypeScript

```
// Message.tsx

import React, { useState } from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

export type Props = {
  message: ChatMessage;
  onDelete(id: string): Promise<void>;
};

export const Message = ({ message onDelete }: Props) => {
  const { userId } = useUserContext();
  const [isDeleting, setIsDeleting] = useState(false);
  const isMine = message.sender.userId === userId;
  const handleDelete = async () => {
    setIsDeleting(true);
    try {
      await onDelete(message.id);
    } catch (e) {
      console.log(e);
      // handle chat error here...
    } finally {
      setIsDeleting(false);
    }
  };

  return (
    <div style={{ backgroundColor: isMine ? 'lightblue' : 'silver', padding: 6,
 borderRadius: 10, margin: 10 }}>
      <p>{content}</p>
```

```
      <button onClick={handleDelete} disabled={isDeleting}>
        Delete
      </button>
    </div>
  );
};
```

JavaScript

```
// Message.jsx

import React, { useState } from 'react';
import { useUserContext } from './UserContext';

export const Message = ({ message, onDelete }) => {
  const { userId } = useUserContext();
  const [isDeleting, setIsDeleting] = useState(false);
  const isMine = message.sender.userId === userId;
  const handleDelete = async () => {
    setIsDeleting(true);
    try {
      await onDelete(message.id);
    } catch (e) {
      console.log(e);
      // handle the exceptions here...
    } finally {
      setIsDeleting(false);
    }
  };

  return (
    <div style={{ backgroundColor: 'silver', padding: 6, borderRadius: 10, margin:
 10 }}>
      <p>{message.content}</p>
      <button onClick={handleDelete} disabled={isDeleting}>
        Delete
      </button>
    </div>
  );
};
```

Selanjutnya, Anda memperbarui `MessageList` untuk merefleksikan perubahan terbaru pada komponen `Message` Anda.

Buka `MessageList` dan tentukan fungsi baru yang disebut `onDelete`, yang menerima string sebagai parameter dan mengembalikan `Promise`. Perbarui `Message` Anda dan teruskan melalui properti `Message`. Parameter string dalam penutupan baru Anda akan menjadi ID pesan yang ingin dihapus, yang diteruskan dari `Message` Anda.

TypeScript

```
// MessageList.tsx

import * as React from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { Message } from './Message';

interface Props {
  messages: ChatMessage[];
  onDelete(id: string): Promise<void>;
}

export const MessageList = ({ messages, onDelete }: Props) => {
  return (
    <>
      {messages.map((message) => (
        <Message key={message.id} onDelete={onDelete} content={message.content}
 id={message.id} />
      ))}
    </>
  );
};
```

JavaScript

```
// MessageList.jsx

import * as React from 'react';
import { Message } from './Message';

export const MessageList = ({ messages, onDelete }) => {
  return (
    <>
      {messages.map((message) => (
```

```
        <Message key={message.id} onDelete={onDelete} content={message.content}
 id={message.id} />
      ))}
    </>
  );
};
```

Selanjutnya, Anda memperbarui App untuk menunjukkan perubahan terbaru pada `MessageList` Anda.

Di App, tentukan fungsi bernama `onDeleteMessage` dan teruskan ke properti `MessageList` `onDelete`:

TypeScript

```
// App.tsx

// ...

const onDeleteMessage = async (id: string) => {};

return (
  <div style={{ display: 'flex', flexDirection: 'column', padding: 10 }}>
    <h4>Connection State: {connectionState}</h4>
    <MessageList onDelete={onDeleteMessage} messages={messages} />
    <div style={{ flexDirection: 'row', display: 'flex', width: '100%' }}>
      <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onSendPress={onMessageSend} />
    </div>
  </div>
);

// ...
```

JavaScript

```
// App.jsx

// ...

const onDeleteMessage = async (id) => {};
```

```
return (
  <div style={{ display: 'flex', flexDirection: 'column', padding: 10 }}>
    <h4>Connection State: {connectionState}</h4>
    <MessageList onDelete={onDeleteMessage} messages={messages} />
    <div style={{ flexDirection: 'row', display: 'flex', width: '100%' }}>
      <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onSendPress={onMessageSend} />
    </div>
  </div>
);

// ...
```

Siapkan permintaan dengan membuat instans `DeleteMessageRequest` baru, dengan meneruskan ID pesan yang relevan ke parameter konstruktor, dan panggil `deleteMessage` yang menerima permintaan yang disiapkan di atas:

TypeScript

```
// App.tsx

// ...

const onDeleteMessage = async (id: string) => {
  const request = new DeleteMessageRequest(id);
  await room.deleteMessage(request);
};

// ...
```

JavaScript

```
// App.jsx

// ...

const onDeleteMessage = async (id) => {
  const request = new DeleteMessageRequest(id);
  await room.deleteMessage(request);
};
```

```
// ...
```

Selanjutnya, Anda memperbarui status `messages` untuk merefleksikan daftar pesan baru yang menghilangkan pesan yang baru saja Anda hapus.

Di hook `useEffect`, dengarkan peristiwa `messageDelete` dan perbarui array status `messages` Anda dengan menghapus pesan yang mempunyai ID yang sama dengan parameter `message`.

Catatan: Peristiwa `messageDelete` dapat dimunculkan untuk pesan yang dihapus oleh pengguna saat ini atau pengguna lain di ruang. Dengan menanganinya di handler peristiwa (bukan di samping permintaan `deleteMessage`), Anda dapat menyatukan penanganan hapus-pesan.

```
// App.jsx / App.tsx

// ...

const unsubscribeOnMessageDeleted = room.addListener('messageDelete',
 (deleteMessageEvent) => {
  setMessages((prev) => prev.filter((message) => message.id !==
 deleteMessageEvent.id));
});

return () => {
  // ...

  unsubscribeOnMessageDeleted();
};

// ...
```

Sekarang Anda dapat menghapus pengguna dari ruang obrolan di aplikasi obrolan Anda.

Langkah Berikutnya

Sebagai percobaan, coba implementasikan tindakan lain di suatu ruang, seperti memutus koneksi pengguna lain.

IVSPesan Klien ObrolanSDK: React Native Tutorial Bagian 1: Ruang Obrolan

Ini adalah bagian pertama dari tutorial dua bagian. Anda akan mempelajari hal-hal penting dari bekerja dengan Amazon IVS Chat Client Messaging JavaScript SDK dengan membangun aplikasi yang berfungsi penuh menggunakan React Native. Kami menyebut aplikasi itu Chatterbox.

Audiens yang dituju adalah pengembang berpengalaman yang baru mengenal Amazon IVS Chat MessagingSDK. Anda harus merasa nyaman dengan TypeScript atau bahasa JavaScript pemrograman dan perpustakaan React Native.

Untuk singkatnya, kita akan merujuk ke Pesan Klien IVS Obrolan Amazon JavaScript SDK sebagai JS SDK Obrolan.

Catatan: Dalam beberapa kasus, contoh kode untuk JavaScript dan TypeScript identik, sehingga digabungkan.

Bagian pertama dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Menyiapkan Server Autentikasi/Otorisasi Lokal”
2. the section called “Buat Proyek Chatterbox”
3. the section called “Menghubungkan ke Ruang Obrolan”
4. the section called “Membangun Penyedia Token”
5. the section called “Mengamati Pembaruan Koneksi”
6. the section called “Membuat Komponen Tombol Kirim”
7. the section called “Buat Input Pesan”
8. the section called “Langkah Berikutnya”

Prasyarat

- Biasakan diri dengan TypeScript atau JavaScript dan library React Native. Jika Anda tidak paham dengan React Native, pelajari dasar-dasarnya di Pengantar React Native.
- Baca dan pahami Memulai dengan IVS Chat.
- Buat AWS IAM pengguna dengan CreateRoom kemampuan CreateChatToken dan yang ditentukan dalam IAM kebijakan yang ada. (Lihat Memulai dengan IVS Chat).

- Pastikan kunci rahasia/akses untuk pengguna ini disimpan dalam file AWS kredensial. Untuk petunjuk, lihat Panduan AWS CLI Pengguna (terutama Konfigurasi dan pengaturan file kredenal).
- Buat ruang obrolan dan simpanARN. Lihat Memulai dengan IVS Chat. (Jika Anda tidak menyimpanARN, Anda dapat mencarinya nanti dengan konsol atau ObrolanAPI.)
- Instal lingkungan Node.js 14+ dengan manajer paket NPM atau Yarn.

Menyiapkan Server Autentikasi/Otorisasi Lokal

Aplikasi backend Anda bertanggung jawab untuk membuat ruang obrolan dan menghasilkan token obrolan yang diperlukan untuk JS SDK Obrolan untuk mengautentikasi dan mengotorisasi klien Anda untuk ruang obrolan Anda. Anda harus menggunakan backend Anda sendiri karena Anda tidak dapat menyimpan AWS kunci dengan aman di aplikasi seluler; penyerang canggih dapat mengekstrak ini dan mendapatkan akses ke akun Anda. AWS

Lihat Membuat Token Obrolan di Memulai IVS Obrolan Amazon. Seperti yang ditunjukkan pada diagram alur di sana, aplikasi sisi server Anda bertanggung jawab untuk membuat token obrolan. Hal ini berarti aplikasi Anda harus menyediakan caranya sendiri untuk menghasilkan token obrolan dengan memintanya dari aplikasi sisi server Anda.

Di bagian ini, Anda akan mempelajari dasar-dasar membuat penyedia token di backend Anda. Kami menggunakan kerangka kerja ekspres untuk membuat server lokal langsung yang mengelola pembuatan token obrolan menggunakan AWS lingkungan lokal Anda.

Buat `npm` proyek kosong menggunakanNPM. Buat direktori untuk menyimpan aplikasi Anda, dan jadikan sebagai direktori kerja Anda:

```
$ mkdir backend & cd backend
```

Gunakan `npm init` untuk membuat file `package.json` pada aplikasi Anda:

```
$ npm init
```

Perintah ini meminta beberapa hal pada Anda, termasuk nama dan versi aplikasi Anda. Untuk saat ini, cukup tekan RETURNuntuk menerima default untuk sebagian besar dari mereka, dengan pengecualian berikut:

```
entry point: (index.js)
```

Tekan RETURNuntuk menerima nama file default yang disarankan `index.js` atau masukkan nama file utama apa pun yang Anda inginkan.

Sekarang instal dependensi yang diperlukan:

```
$ npm install express aws-sdk cors dotenv
```

`aws-sdk` memerlukan variabel lingkungan konfigurasi, yang memuat secara otomatis dari file bernama `.env` yang terletak di direktori root. Untuk mengonfigurasikannya, buat file baru bernama `.env` dan isi informasi konfigurasi yang hilang:

```
# .env

# The region to send service requests to.
AWS_REGION=us-west-2

# Access keys use an access key ID and secret access key
# that you use to sign programmatic requests to AWS.

# AWS access key ID.
AWS_ACCESS_KEY_ID=...

# AWS secret access key.
AWS_SECRET_ACCESS_KEY=...
```

Sekarang kita buat file titik masuk di direktori root dengan nama yang Anda masukkan di atas dalam perintah `npm init`. Dalam hal ini, kami menggunakan `index.js`, dan mengimpor semua paket yang diperlukan:

```
// index.js
import express from 'express';
import AWS from 'aws-sdk';
import 'dotenv/config';
import cors from 'cors';
```

Sekarang, buat instans baru `express`:

```
const app = express();
const port = 3000;
```

```
app.use(express.json());
app.use(cors({ origin: ['http://127.0.0.1:5173'] }));
```

Setelah itu Anda dapat membuat POST metode endpoint pertama Anda untuk penyedia token. Ambil parameter yang diperlukan dari isi permintaan (`roomId`, `userId`, `capabilities`, dan `sessionDurationInMinutes`):

```
app.post('/create_chat_token', (req, res) => {
  const { roomIdentifier, userId, capabilities, sessionDurationInMinutes } = req.body
 || {};
});
```

Tambahkan validasi bidang yang wajib diisi:

```
app.post('/create_chat_token', (req, res) => {
  const { roomIdentifier, userId, capabilities, sessionDurationInMinutes } = req.body
 || {};

  if (!roomIdentifier || !userId) {
    res.status(400).json({ error: 'Missing parameters: `roomIdentifier`, `userId`' });
    return;
  }
});
```

Setelah menyiapkan POST metode, kami mengintegrasikan `createChatToken` dengan `aws-sdk` fungsionalitas inti otentikasi/otorisasi:

```
app.post('/create_chat_token', (req, res) => {
  const { roomIdentifier, userId, capabilities, sessionDurationInMinutes } = req.body
 || {};

  if (!roomIdentifier || !userId || !capabilities) {
    res.status(400).json({ error: 'Missing parameters: `roomIdentifier`, `userId`,
 `capabilities`' });
    return;
  }

  ivsChat.createChatToken({ roomIdentifier, userId, capabilities,
 sessionDurationInMinutes }, (error, data) => {
    if (error) {
      console.log(error);
```

```
      res.status(500).send(error.code);
    } else if (data.token) {
      const { token, sessionExpirationTime, tokenExpirationTime } = data;
      console.log(`Retrieved Chat Token: ${JSON.stringify(data, null, 2)}`);

      res.json({ token, sessionExpirationTime, tokenExpirationTime });
    }
  });
});
```

Di akhir file, tambahkan pendengar port untuk aplikasi `express` Anda:

```
app.listen(port, () => {
  console.log(`Backend listening on port ${port}`);
});
```

Sekarang Anda dapat menjalankan server dengan perintah berikut dari root proyek:

```
$ node index.js
```

Tip: Server ini menerima URL permintaan di https://localhost:3000.

Buat Proyek Chatterbox

Pertama, Anda membuat proyek React Native yang disebut `chatterbox`. Jalankan perintah ini:

```
npx create-expo-app
```

Atau buat proyek expo dengan TypeScript template.

```
npx create-expo-app -t expo-template-blank-typescript
```

Anda dapat mengintegrasikan Chat Client Messaging JS SDK melalui Node Package Manager atau Yarn Package Manager:

- Npm: `npm install amazon-ivs-chat-messaging`
- Yarn: `yarn add amazon-ivs-chat-messaging`

Menghubungkan ke Ruang Obrolan

Di sini Anda membuat `ChatRoom` dan menghubungkannya menggunakan metode asinkron. `ChatRoom`Kelas mengelola koneksi pengguna Anda ke Chat JSSDK. Agar berhasil terhubung ke ruang obrolan, Anda harus menyediakan instans `ChatToken` dalam aplikasi React Anda.

Arahkan ke file App yang dibuat dalam proyek `chatterbox` default dan hapus semua yang dikembalikan oleh komponen fungsional. Tidak memerlukan kode yang telah diisi sebelumnya. Saat ini, App kami cukup kosong.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

import * as React from 'react';
import { Text } from 'react-native';

export default function App() {
  return <Text>Hello!</Text>;
}
```

Buat instans `ChatRoom` baru dan teruskan ke status menggunakan hook `useState`. Ini membutuhkan passing `regionOrUrl` (AWSwilayah di mana ruang obrolan Anda di-host) dan `tokenProvider` (digunakan untuk aliran otentikasi/otorisasi backend yang dibuat pada langkah selanjutnya).

Penting: Anda harus menggunakan AWS wilayah yang sama dengan wilayah tempat Anda membuat ruangan di Memulai dengan IVS Obrolan Amazon. APIIni adalah layanan AWS regional. Untuk daftar wilayah yang didukung dan titik akhir HTTPS layanan IVS Obrolan Amazon, lihat halaman wilayah IVS Obrolan Amazon.

TypeScript/JavaScript:

```
// App.jsx / App.tsx

import React, { useState } from 'react';
import { Text } from 'react-native';
import { ChatRoom } from 'amazon-ivs-chat-messaging';

export default function App() {
  const [room] = useState(() =>
```

```
    new ChatRoom({
      regionOrUrl: process.env.REGION,
      tokenProvider: () => {},
    }),
  );

  return <Text>Hello!</Text>;
}
```

Membangun Penyedia Token

Sebagai langkah berikutnya, kita perlu membangun fungsi `tokenProvider` tanpa parameter yang diperlukan oleh konstruktor `ChatRoom`. Pertama, kita akan membuat `fetchChatToken` fungsi yang akan membuat POST permintaan ke aplikasi backend yang Anda atur. the section called "Menyiapkan Server Autentikasi/Otorisasi Lokal" Token obrolan berisi informasi yang diperlukan SDK agar berhasil membuat koneksi ruang obrolan. Obrolan API menggunakan token ini sebagai cara aman untuk memvalidasi identitas pengguna, kemampuan dalam ruang obrolan, dan durasi sesi.

Di navigator Proyek, buat JavaScript file TypeScript/baru bernama`fetchChatToken`. Bangun permintaan pengambilan ke aplikasi `backend` dan kembalikan objek `ChatToken` dari respons. Tambahkan properti isi permintaan yang diperlukan untuk membuat token obrolan. Gunakan aturan yang ditentukan untuk Amazon Resource Names (ARNs). Properti ini didokumentasikan di CreateChatToken titik akhir.

Catatan: Yang URL Anda gunakan di sini sama dengan server lokal Anda URL yang dibuat ketika Anda menjalankan aplikasi backend.

TypeScript

```
// fetchChatToken.ts

import { ChatToken } from 'amazon-ivs-chat-messaging';

type UserCapability = 'DELETE_MESSAGE' | 'DISCONNECT_USER' | 'SEND_MESSAGE';

export async function fetchChatToken(
  userId: string,
  capabilities: UserCapability[] = [],
  attributes?: Record<string, string>,
  sessionDurationInMinutes?: number,
): Promise<ChatToken> {
```

```
  const response = await fetch(`${process.env.BACKEND_BASE_URL}/create_chat_token`,
 {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      userId,
      roomIdentifier: process.env.ROOM_ID,
      capabilities,
      sessionDurationInMinutes,
      attributes
    }),
  });

  const token = await response.json();

  return {
    ...token,
    sessionExpirationTime: new Date(token.sessionExpirationTime),
    tokenExpirationTime: new Date(token.tokenExpirationTime),
  };
}
```

JavaScript

```
// fetchChatToken.js

export async function fetchChatToken(
  userId,
  capabilities = [],
  attributes,
  sessionDurationInMinutes) {
  const response = await fetch(`${process.env.BACKEND_BASE_URL}/create_chat_token`,
 {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      userId,
```

```
      roomIdentifier: process.env.ROOM_ID,
      capabilities,
      sessionDurationInMinutes,
      attributes
    }),
  });

  const token = await response.json();

  return {
    ...token,
    sessionExpirationTime: new Date(token.sessionExpirationTime),
    tokenExpirationTime: new Date(token.tokenExpirationTime),
  };
}
```

Mengamati Pembaruan Koneksi

Bereaksi terhadap perubahan status koneksi ruang obrolan adalah bagian penting dalam membuat aplikasi obrolan. Mari kita mulai dengan berlangganan peristiwa yang relevan:

TypeScript/JavaScript:

```
// App.tsx / App.jsx

import React, { useState, useEffect } from 'react';
import { Text } from 'react-native';
import { ChatRoom } from 'amazon-ivs-chat-messaging';
import { fetchChatToken } from './fetchChatToken';

export default function App() {
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION,
        tokenProvider: () => fetchChatToken('Mike', ['SEND_MESSAGE']),
      }),
  );

  useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {});
    const unsubscribeOnConnected = room.addListener('connect', () => {});
```

```
    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {});

    return () => {
      // Clean up subscriptions.
      unsubscribeOnConnecting();
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, [room]);

  return <Text>Hello!</Text>;
}
```

Selanjutnya, kita perlu menyediakan kemampuan untuk membaca status koneksi. Kita menggunakan hook `useState` untuk membuat beberapa status lokal di `App` dan mengatur status koneksi di dalam setiap pendengar.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

import React, { useState, useEffect } from 'react';
import { Text } from 'react-native';
import { ChatRoom, ConnectionState } from 'amazon-ivs-chat-messaging';
import { fetchChatToken } from './fetchChatToken';

export default function App() {
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION,
        tokenProvider: () => fetchChatToken('Mike', ['SEND_MESSAGE']),
      }),
  );
  const [connectionState, setConnectionState] =
 useState<ConnectionState>('disconnected');

  useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {
      setConnectionState('connecting');
    });

    const unsubscribeOnConnected = room.addListener('connect', () => {
```

```
      setConnectionState('connected');
    });

    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
      setConnectionState('disconnected');
    });

    return () => {
      unsubscribeOnConnecting();
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, [room]);

  return <Text>Hello!</Text>;
}
```

Setelah berlangganan status koneksi, tampilkan status koneksi dan hubungkan ke ruang obrolan dengan menggunakan metode `room.connect` di dalam hook `useEffect`:

TypeScript/JavaScript:

```
// App.tsx / App.jsx

// ...

useEffect(() => {
  const unsubscribeOnConnecting = room.addListener('connecting', () => {
    setConnectionState('connecting');
  });

  const unsubscribeOnConnected = room.addListener('connect', () => {
    setConnectionState('connected');
  });

  const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
    setConnectionState('disconnected');
  });

  room.connect();

  return () => {
    unsubscribeOnConnecting();
```

```
    unsubscribeOnConnected();
    unsubscribeOnDisconnected();
  };
}, [room]);

// ...

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
  </SafeAreaView>
);

const styles = StyleSheet.create({
  root: {
    flex: 1,
  }
});

// ...
```

Anda telah berhasil mengimplementasikan koneksi ruang obrolan.

Membuat Komponen Tombol Kirim

Di bagian ini, Anda membuat tombol kirim yang memiliki desain berbeda untuk setiap status koneksi. Tombol kirim memfasilitasi pengiriman pesan di ruang obrolan. Tombol ini juga berfungsi sebagai indikator visual mengenai apakah/kapan pesan dapat dikirim; misalnya, saat koneksi terputus atau sesi obrolan yang kedaluwarsa.

Pertama, buat file baru di direktori `src` proyek Chatterbox Anda dan beri nama `SendButton`. Kemudian, buat komponen yang akan menampilkan tombol untuk aplikasi obrolan Anda. Ekspor `SendButton` Anda dan impor ke `App`. Di `<View></View>` yang kosong, tambahkan `<SendButton />`.

TypeScript

```
// SendButton.tsx

import React from 'react';
import { TouchableOpacity, Text, ActivityIndicator, StyleSheet } from 'react-
native';
```

```
interface Props {
  onPress?: () => void;
  disabled: boolean;
  loading: boolean;
}

export const SendButton = ({ onPress, disabled, loading }: Props) => {
  return (
    <TouchableOpacity style={styles.root} disabled={disabled} onPress={onPress}>
      {loading ? <Text>Send</Text> : <ActivityIndicator />}
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  root: {
    width: 50,
    height: 50,
    borderRadius: 30,
    marginLeft: 10,
    justifyContent: 'center',
    alignContent: 'center',
  }
});

// App.tsx

import { SendButton } from './SendButton';

// ...

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <SendButton />
  </SafeAreaView>
);
```

JavaScript

```
// SendButton.jsx
```

```
import React from 'react';
import { TouchableOpacity, Text, ActivityIndicator, StyleSheet } from 'react-
native';

export const SendButton = ({ onPress, disabled, loading }) => {
  return (
    <TouchableOpacity style={styles.root} disabled={disabled} onPress={onPress}>
      {loading ? <Text>Send</Text> : <ActivityIndicator />}
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  root: {
    width: 50,
    height: 50,
    borderRadius: 30,
    marginLeft: 10,
    justifyContent: 'center',
    alignContent: 'center',
  }
});

// App.jsx

import { SendButton } from './SendButton';

// ...

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <SendButton />
  </SafeAreaView>
);
```

Selanjutnya di App, tentukan fungsi bernama `onMessageSend` dan berikan fungsi tersebut ke properti `SendButton onPress`. Tentukan variabel lain bernama `isSendDisabled` (yang mencegah pengiriman pesan ketika ruang tidak terhubung) dan berikan variabel tersebut ke properti `SendButton disabled`.

TypeScript/JavaScript:

```
// App.jsx / App.tsx

// ...

const onMessageSend = () => {};

const isSendDisabled = connectionState !== 'connected';

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
  </SafeAreaView>
);

// ...
```

Buat Input Pesan

Bilah pesan Chatterbox adalah komponen yang akan berinteraksi dengan Anda untuk mengirim pesan ke ruang obrolan. Biasanya, bilah ini berisi input teks untuk menulis pesan dan tombol untuk mengirim pesan Anda.

Untuk membuat komponen `MessageInput`, pertama buat file baru di direktori `src` dan beri nama `MessageInput`. Kemudian, buat komponen yang akan menampilkan input untuk aplikasi obrolan Anda. Ekspor `MessageInput` Anda dan impor ke `App` (di atas `<SendButton />`).

Buat status baru bernama `messageToSend` menggunakan hook `useState`, dengan string kosong sebagai nilai default-nya. Di bagian utama aplikasi Anda, berikan `messageToSend` ke `value` dari `MessageInput` dan berikan `setMessageToSend` ke properti `onMessageChange`:

TypeScript

```
// MessageInput.tsx

import * as React from 'react';

interface Props {
  value?: string;
  onValueChange?: (value: string) => void;
}
```

```
export const MessageInput = ({ value, onValueChange }: Props) => {
  return (
    <TextInput style={styles.input} value={value} onChangeText={onValueChange}
 placeholder="Send a message" />
  );
};

const styles = StyleSheet.create({
  input: {
    fontSize: 20,
    backgroundColor: 'rgb(239,239,240)',
    paddingHorizontal: 18,
    paddingVertical: 15,
    borderRadius: 50,
    flex: 1,
  }
})

// App.tsx

// ...

import { MessageInput } from './MessageInput';

// ...

export default function App() {
  const [messageToSend, setMessageToSend] = useState('');

// ...

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <View style={styles.messageBar}>
      <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
    </View>
  </SafeAreaView>
);

const styles = StyleSheet.create({
  root: {
    flex: 1,
```

```
  },
  messageBar: {
    borderTopWidth: StyleSheet.hairlineWidth,
    borderTopColor: 'rgb(160,160,160)',
    flexDirection: 'row',
    padding: 16,
    alignItems: 'center',
    backgroundColor: 'white',
  }
});
```

JavaScript

```
// MessageInput.jsx

import * as React from 'react';

export const MessageInput = ({ value, onValueChange }) => {
  return (
    <TextInput style={styles.input} value={value} onChangeText={onValueChange}
 placeholder="Send a message" />
  );
};

const styles = StyleSheet.create({
  input: {
    fontSize: 20,
    backgroundColor: 'rgb(239,239,240)',
    paddingHorizontal: 18,
    paddingVertical: 15,
    borderRadius: 50,
    flex: 1,
  }
})

// App.jsx

// ...

import { MessageInput } from './MessageInput';

// ...
```

```
export default function App() {
  const [messageToSend, setMessageToSend] = useState('');

// ...

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <View style={styles.messageBar}>
      <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
    </View>
  </SafeAreaView>
);

const styles = StyleSheet.create({
  root: {
    flex: 1,
  },
  messageBar: {
    borderTopWidth: StyleSheet.hairlineWidth,
    borderTopColor: 'rgb(160,160,160)',
    flexDirection: 'row',
    padding: 16,
    alignItems: 'center',
    backgroundColor: 'white',
  }
});
```

Langkah Berikutnya

Karena Anda telah selesai membuat bilah pesan untuk Chatterbox, lanjutkan ke Bagian 2 tutorial React Native, Pesan dan Peristiwa.

IVSPesan Klien ObrolanSDK: React Native Tutorial Bagian 2: Pesan dan Acara

Bagian kedua (dan terakhir) dari tutorial ini dipecah menjadi beberapa bagian:

1. the section called “Berlangganan Peristiwa Pesan Obrolan”

2. the section called “Menampilkan Pesan yang Diterima”
   a. the section called “Membuat Komponen Pesan”
   b. the section called “Mengenali Pesan yang Dikirim oleh Pengguna Saat Ini”
   c. the section called “Me-render Daftar Pesan Obrolan”
3. the section called “Melakukan Tindakan di Ruang Obrolan”
   a. the section called “Mengirim Pesan”
   b. the section called “Menghapus Pesan”
4. the section called “Langkah Berikutnya”

Catatan: Dalam beberapa kasus, contoh kode untuk JavaScript dan TypeScript identik, sehingga digabungkan.

Prasyarat

Pastikan Anda telah menyelesaikan Bagian 1 dari tutorial ini, Ruang Obrolan.

Berlangganan Peristiwa Pesan Obrolan

Instans `ChatRoom` menggunakan peristiwa untuk berkomunikasi ketika peristiwa terjadi di ruang obrolan. Untuk mulai mengimplementasikan pengalaman obrolan, Anda harus menunjukkan kepada pengguna saat orang lain mengirim pesan di ruang yang terhubung dengan mereka.

Di sini, Anda berlangganan peristiwa pesan obrolan. Selanjutnya, kami akan menunjukkan cara memperbarui daftar pesan yang Anda buat, yang diperbarui dengan setiap pesan/peristiwa.

Di `App` Anda, di dalam hook `useEffect`, berlangganan semua peristiwa pesan:

TypeScript/JavaScript:

```
// App.tsx / App.jsx

useEffect(() => {
  // ...
  const unsubscribeOnMessageReceived = room.addListener('message', (message) => {});

  return () => {
    // ...
    unsubscribeOnMessageReceived();
  };
```

```
}, []);
```

Menampilkan Pesan yang Diterima

Menerima pesan adalah bagian inti dari pengalaman obrolan. Menggunakan Chat JSSDK, Anda dapat mengatur kode Anda untuk dengan mudah menerima acara dari pengguna lain yang terhubung ke ruang obrolan.

Selanjutnya, kami akan menunjukkan cara melakukan tindakan di ruang obrolan yang memanfaatkan komponen yang Anda buat di sini.

Di App Anda, tentukan status bernama `messages` dengan tipe array `ChatMessage` yang bernama `messages`:

TypeScript

```
// App.tsx

// ...

import { ChatRoom, ChatMessage, ConnectionState } from 'amazon-ivs-chat-messaging';

export default function App() {
  const [messages, setMessages] = useState<ChatMessage[]>([]);

  //...
}
```

JavaScript

```
// App.jsx

// ...

import { ChatRoom, ConnectionState } from 'amazon-ivs-chat-messaging';

export default function App() {
  const [messages, setMessages] = useState([]);

  //...
}
```

Selanjutnya, di fungsi pendengar `message`, tambahkan `message` ke array `messages`:

TypeScript/JavaScript:

```
// App.tsx / App.jsx

// ...

const unsubscribeOnMessageReceived = room.addListener('message', (message) => {
  setMessages((msgs) => [...msgs, message]);
});

// ...
```

Di bawah ini, kita akan menjalankan langkah demi langkah untuk menampilkan pesan yang diterima:

1. the section called “Membuat Komponen Pesan”
2. the section called “Mengenali Pesan yang Dikirim oleh Pengguna Saat Ini”
3. the section called “Me-render Daftar Pesan Obrolan”

Membuat Komponen Pesan

Komponen `Message` bertanggung jawab untuk me-render konten pesan yang diterima oleh ruang obrolan Anda. Di bagian ini, Anda membuat komponen pesan untuk me-render pesan obrolan individu di `App`.

Buat file baru di direktori `src` dan beri nama `Message`. Berikan tipe `ChatMessage` untuk komponen ini, dan berikan string `content` dari properti `ChatMessage` untuk menampilkan teks pesan yang diterima dari pendengar pesan ruang obrolan. Di Navigator Proyek, buka `Message`.

TypeScript

```
// Message.tsx

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { ChatMessage } from 'amazon-ivs-chat-messaging';

type Props = {
  message: ChatMessage;
```

```
}

export const Message = ({ message }: Props) => {
  return (
    <View style={styles.root}>
      <Text>{message.sender.userId}</Text>
      <Text style={styles.textContent}>{message.content}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  root: {
    backgroundColor: 'silver',
    padding: 6,
    borderRadius: 10,
    marginHorizontal: 12,
    marginVertical: 5,
    marginRight: 50,
  },
  textContent: {
    fontSize: 17,
    fontWeight: '500',
    flexShrink: 1,
  },
});
```

JavaScript

```
// Message.jsx

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

export const Message = ({ message }) => {
  return (
    <View style={styles.root}>
      <Text>{message.sender.userId}</Text>
      <Text style={styles.textContent}>{message.content}</Text>
    </View>
  );
};
```

```
const styles = StyleSheet.create({
  root: {
    backgroundColor: 'silver',
    padding: 6,
    borderRadius: 10,
    marginHorizontal: 12,
    marginVertical: 5,
    marginRight: 50,
  },
  textContent: {
    fontSize: 17,
    fontWeight: '500',
    flexShrink: 1,
  },
});
```

Tip: Gunakan komponen ini untuk menyimpan properti berbeda yang ingin Anda render di baris pesan; misalnya, avatarURLs, nama pengguna, dan stempel waktu saat pesan dikirim.

Mengenali Pesan yang Dikirim oleh Pengguna Saat Ini

Untuk mengenali pesan yang dikirim oleh pengguna saat ini, kita mengubah kode dan membuat konteks React untuk menyimpan `userId` pengguna saat ini.

Buat file baru di direktori `src` dan beri nama `UserContext`:

TypeScript

```
// UserContext.tsx

import React from 'react';

const UserContext = React.createContext<string | undefined>(undefined);

export const useUserContext = () => {
  const context = React.useContext(UserContext);

  if (context === undefined) {
    throw new Error('useUserContext must be within UserProvider');
  }

  return context;
```

```
};

export const UserProvider = UserContext.Provider;
```

JavaScript

```
// UserContext.jsx

import React from 'react';

const UserContext = React.createContext(undefined);

export const useUserContext = () => {
  const context = React.useContext(UserContext);

  if (context === undefined) {
    throw new Error('useUserContext must be within UserProvider');
  }

  return context;
};

export const UserProvider = UserContext.Provider;
```

Catatan: Di sini kita menggunakan hook `useState` untuk menyimpan nilai `userId`. Ke depannya, Anda dapat memanfaatkan `setUserId` untuk mengubah konteks pengguna atau untuk tujuan login.

Selanjutnya, ganti `userId` pada parameter pertama yang diberikan ke `tokenProvider`, dengan menggunakan konteks yang dibuat sebelumnya. Pastikan untuk menambahkan kemampuan `SEND_MESSAGE` ke penyedia token Anda, seperti yang ditentukan di bawah ini; kemampuan ini diperlukan untuk mengirim pesan.:

TypeScript

```
// App.tsx

// ...

import { useUserContext } from './UserContext';

// ...
```

```
export default function App() {
  const [messages, setMessages] = useState<ChatMessage[]>([]);
  const userId = useUserContext();
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION,
        tokenProvider: () => tokenProvider(userId, ['SEND_MESSAGE']),
      }),
  );

  // ...
}
```

JavaScript

```
// App.jsx

// ...

import { useUserContext } from './UserContext';

// ...


export default function App() {
  const [messages, setMessages] = useState([]);
  const userId = useUserContext();
  const [room] = useState(
    () =>
      new ChatRoom({
        regionOrUrl: process.env.REGION,
        tokenProvider: () => tokenProvider(userId, ['SEND_MESSAGE']),
      }),
  );

  // ...
}
```

Dalam komponen `Message` Anda, gunakan `UserContext` yang dibuat sebelumnya, nyatakan variabel `isMine`, cocokkan `userId` pengirim dengan `userId` dari konteks, dan terapkan gaya pesan yang berbeda untuk pengguna saat ini.

TypeScript

```
// Message.tsx

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

type Props = {
  message: ChatMessage;
}

export const Message = ({ message }: Props) => {
  const userId = useUserContext();

  const isMine = message.sender.userId === userId;

  return (
    <View style={[styles.root, isMine && styles.mine]}>
      {!isMine && <Text>{message.sender.userId}</Text>}
      <Text style={styles.textContent}>{message.content}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  root: {
    backgroundColor: 'silver',
    padding: 6,
    borderRadius: 10,
    marginHorizontal: 12,
    marginVertical: 5,
    marginRight: 50,
  },
  textContent: {
    fontSize: 17,
    fontWeight: '500',
    flexShrink: 1,
```

```
  },
  mine: {
    flexDirection: 'row-reverse',
    backgroundColor: 'lightblue',
  },
});
```

JavaScript

```
// Message.jsx

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

export const Message = ({ message }) => {
  const userId = useUserContext();

  const isMine = message.sender.userId === userId;

  return (
    <View style={[styles.root, isMine && styles.mine]}>
      {!isMine && <Text>{message.sender.userId}</Text>}
      <Text style={styles.textContent}>{message.content}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  root: {
    backgroundColor: 'silver',
    padding: 6,
    borderRadius: 10,
    marginHorizontal: 12,
    marginVertical: 5,
    marginRight: 50,
  },
  textContent: {
    fontSize: 17,
    fontWeight: '500',
    flexShrink: 1,
  },
```

```
  mine: {
    flexDirection: 'row-reverse',
    backgroundColor: 'lightblue',
  },
});
```

Me-render Daftar Pesan Obrolan

Sekarang, buat daftar pesan dengan memanfaatkan komponen `FlatList` dan `Message`:

TypeScript

```
// App.tsx

// ...

const renderItem = useCallback<ListRenderItem<ChatMessage>>(({ item }) => {
  return (
    <Message key={item.id} message={item} />
  );
}, []);

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <FlatList inverted data={messages} renderItem={renderItem} />
    <View style={styles.messageBar}>
      <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
    </View>
  </SafeAreaView>
);

// ...
```

JavaScript

```
// App.jsx

// ...

const renderItem = useCallback(({ item }) => {
```

```
  return (
    <Message key={item.id} message={item} />
  );
}, []);

return (
  <SafeAreaView style={styles.root}>
    <Text>Connection State: {connectionState}</Text>
    <FlatList inverted data={messages} renderItem={renderItem} />
    <View style={styles.messageBar}>
      <MessageInput value={messageToSend} onMessageChange={setMessageToSend} />
      <SendButton disabled={isSendDisabled} onPress={onMessageSend} />
    </View>
  </SafeAreaView>
);

// ...
```

Semua potongan puzzle sekarang sudah siap sehingga App Anda dapat mulai me-render pesan yang diterima oleh ruang obrolan Anda. Lanjutkan langkah di bawah ini untuk melihat cara melakukan tindakan di ruang obrolan yang memanfaatkan komponen yang telah Anda buat.

Melakukan Tindakan di Ruang Obrolan

Mengirim pesan dan melakukan tindakan moderator adalah beberapa cara utama Anda dalam berinteraksi dengan ruang obrolan. Di sini, Anda akan belajar cara menggunakan berbagai objek permintaan obrolan untuk melakukan tindakan umum di Chatterbox, seperti mengirim pesan, menghapus pesan, dan memutus koneksi pengguna lain.

Semua tindakan di ruang obrolan mengikuti pola umum: untuk setiap tindakan yang Anda lakukan di ruang obrolan, ada objek permintaan yang sesuai. Untuk setiap permintaan, ada objek respons yang sesuai yang Anda terima setelah konfirmasi permintaan.

Selama pengguna Anda diberi kemampuan yang benar saat Anda membuat token obrolan, mereka akan berhasil melakukan tindakan yang sesuai menggunakan objek permintaan untuk melihat permintaan apa yang dapat Anda lakukan di ruang obrolan.

Di bawah ini, kami menjelaskan cara mengirim pesan dan menghapus pesan.

Mengirim Pesan

Kelas `SendMessageRequest` memungkinkan pengiriman pesan di ruang obrolan. Di sini, Anda memodifikasi App untuk mengirim permintaan pesan menggunakan komponen yang Anda buat di Buat Input Pesan (di Bagian 1 tutorial ini).

Untuk memulai, tentukan properti boolean baru bernama `isSending` dengan hook `useState`. Gunakan properti baru ini untuk mengalihkan status nonaktif elemen `button` Anda, menggunakan konstanta `isSendDisabled`. Di handler peristiwa untuk `SendButton` Anda, kosongkan nilai untuk `messageToSend` dan atur `isSending` ke true.

Karena Anda akan melakukan API panggilan dari tombol ini, menambahkan *`isSending`* boolean membantu mencegah beberapa API panggilan terjadi pada saat yang sama, dengan menonaktifkan interaksi pengguna pada Anda *`SendButton`* hingga permintaan selesai.

Catatan: Pengiriman pesan hanya akan berhasil jika Anda menambahkan kemampuan `SEND_MESSAGE` ke penyedia token, seperti yang dibahas di atas dalam Mengenali Pesan yang Dikirim oleh Pengguna Saat ini.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

// ...

const [isSending, setIsSending] = useState(false);

// ...

const onMessageSend = () => {
  setIsSending(true);
  setMessageToSend('');
};

// ...

const isSendDisabled = connectionState !== 'connected' || isSending;

// ...
```

Siapkan permintaan dengan membuat instans `SendMessageRequest` baru, dengan meneruskan konten pesan ke konstruktor. Setelah mengatur status `isSending` dan `messageToSend`, panggil

metode `sendMessage`, yang mengirimkan permintaan ke ruang obrolan. Terakhir, hapus bendera `isSending` saat menerima konfirmasi atau penolakan permintaan.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

// ...
import { ChatRoom, ConnectionState, SendMessageRequest } from 'amazon-ivs-chat-
messaging'
// ...

const onMessageSend = async () => {
  const request = new SendMessageRequest(messageToSend);
  setIsSending(true);
  setMessageToSend('');

  try {
    const response = await room.sendMessage(request);
  } catch (e) {
    console.log(e);
    // handle the chat error here...
  } finally {
    setIsSending(false);
  }
};

// ...
```

Jalankan Chatterbox: coba mengirim pesan dengan menyusun pesan menggunakan `MessageBar` Anda dan mengetuk `SendButton` Anda. Anda akan melihat pesan terkirim Anda di-render dalam `MessageList` yang dibuat sebelumnya.

Menghapus Pesan

Untuk menghapus pesan dari ruang obrolan, Anda harus memiliki kemampuan yang tepat. Kemampuan diberikan selama inisialisasi token obrolan yang Anda gunakan saat mengautentikasi ke ruang obrolan. Untuk keperluan bagian ini, `ServerApp` dari Menyiapkan Server Autentikasi/Otorisasi Lokal (di Bagian 1 tutorial ini) memungkinkan Anda menentukan kemampuan moderator. Hal ini dilakukan di aplikasi Anda menggunakan objek `tokenProvider` yang Anda buat di Membangun Penyedia Token (juga di Bagian 1).

Di sini, Anda memodifikasi `Message` dengan menambahkan fungsi untuk menghapus pesan.

Pertama, buka `App.tsx` dan tambahkan kemampuan `DELETE_MESSAGE`. (`capabilities` adalah parameter kedua dari fungsi `tokenProvider` Anda.)

Catatan: Ini adalah cara Anda `ServerApp` menginformasikan IVS Obrolan APIs bahwa pengguna yang dikaitkan dengan token obrolan yang dihasilkan dapat menghapus pesan di ruang obrolan. Dalam situasi dunia nyata, Anda mungkin akan memiliki logika backend yang lebih kompleks untuk mengelola kemampuan pengguna di infrastruktur aplikasi server Anda.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

// ...

const [room] = useState(() =>
    new ChatRoom({
      regionOrUrl: process.env.REGION,
      tokenProvider: () => tokenProvider(userId, ['SEND_MESSAGE', 'DELETE_MESSAGE']),
    }),
);

// ...
```

Pada langkah berikutnya, Anda memperbarui `Message` untuk menampilkan tombol hapus.

Tentukan fungsi baru bernama `onDelete`, yang menerima string sebagai salah satu parameternya dan mengembalikan `Promise`. Untuk parameter string, berikan ID pesan komponen Anda.

TypeScript

```
// Message.tsx

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

export type Props = {
  message: ChatMessage;
  onDelete(id: string): Promise<void>;
};
```

```
export const Message = ({ message, onDelete }: Props) => {
  const userId = useUserContext();

  const isMine = message.sender.userId === userId;
  const handleDelete = () => onDelete(message.id);

  return (
    <View style={[styles.root, isMine && styles.mine]}>
      {!isMine && <Text>{message.sender.userId}</Text>}
      <View style={styles.content}>
        <Text style={styles.textContent}>{message.content}</Text>
        <TouchableOpacity onPress={handleDelete}>
          <Text>Delete<Text/>
        </TouchableOpacity>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  root: {
    backgroundColor: 'silver',
    padding: 6,
    borderRadius: 10,
    marginHorizontal: 12,
    marginVertical: 5,
    marginRight: 50,
  },
  content: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  textContent: {
    fontSize: 17,
    fontWeight: '500',
    flexShrink: 1,
  },
  mine: {
    flexDirection: 'row-reverse',
    backgroundColor: 'lightblue',
  },
```

```
});
```

JavaScript

```
// Message.jsx

import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useUserContext } from './UserContext';

export const Message = ({ message, onDelete }) => {
  const userId = useUserContext();

  const isMine = message.sender.userId === userId;
  const handleDelete = () => onDelete(message.id);

  return (
    <View style={[styles.root, isMine && styles.mine]}>
      {!isMine && <Text>{message.sender.userId}</Text>}
      <View style={styles.content}>
        <Text style={styles.textContent}>{message.content}</Text>
        <TouchableOpacity onPress={handleDelete}>
          <Text>Delete<Text/>
        </TouchableOpacity>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  root: {
    backgroundColor: 'silver',
    padding: 6,
    borderRadius: 10,
    marginHorizontal: 12,
    marginVertical: 5,
    marginRight: 50,
  },
  content: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
```

```
  },
  textContent: {
    fontSize: 17,
    fontWeight: '500',
    flexShrink: 1,
  },
  mine: {
    flexDirection: 'row-reverse',
    backgroundColor: 'lightblue',
  },
});
```

Selanjutnya, Anda memperbarui `renderItem` untuk merefleksikan perubahan terbaru pada komponen `FlatList` Anda.

Di App, tentukan fungsi bernama `handleDeleteMessage` dan teruskan ke properti `MessageList` `onDelete`:

TypeScript

```
// App.tsx

// ...

const handleDeleteMessage = async (id: string) => {};

const renderItem = useCallback<ListRenderItem<ChatMessage>>(({ item }) => {
  return (
    <Message key={item.id} message={item} onDelete={handleDeleteMessage} />
  );
}, [handleDeleteMessage]);

// ...
```

JavaScript

```
// App.jsx

// ...

const handleDeleteMessage = async (id) => {};
```

```
const renderItem = useCallback(({ item }) => {
  return (
    <Message key={item.id} message={item} onDelete={handleDeleteMessage} />
  );
}, [handleDeleteMessage]);

// ...
```

Siapkan permintaan dengan membuat instans `DeleteMessageRequest` baru, dengan meneruskan ID pesan yang relevan ke parameter konstruktor, dan panggil `deleteMessage` yang menerima permintaan yang disiapkan di atas:

TypeScript

```
// App.tsx

// ...

const handleDeleteMessage = async (id: string) => {
  const request = new DeleteMessageRequest(id);
  await room.deleteMessage(request);
};

// ...
```

JavaScript

```
// App.jsx

// ...

const handleDeleteMessage = async (id) => {
  const request = new DeleteMessageRequest(id);
  await room.deleteMessage(request);
};

// ...
```

Selanjutnya, Anda memperbarui status `messages` untuk merefleksikan daftar pesan baru yang menghilangkan pesan yang baru saja Anda hapus.

Di hook `useEffect`, dengarkan peristiwa `messageDelete` dan perbarui array status `messages` Anda dengan menghapus pesan yang mempunyai ID yang sama dengan parameter `message`.

Catatan: Peristiwa `messageDelete` dapat dimunculkan untuk pesan yang dihapus oleh pengguna saat ini atau pengguna lain di ruang. Dengan menanganinya di handler peristiwa (bukan di samping permintaan `deleteMessage`), Anda dapat menyatukan penanganan hapus-pesan.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

// ...

const unsubscribeOnMessageDeleted = room.addListener('messageDelete',
 (deleteMessageEvent) => {
  setMessages((prev) => prev.filter((message) => message.id !==
 deleteMessageEvent.id));
});

return () => {
  // ...

  unsubscribeOnMessageDeleted();
};

// ...
```

Sekarang Anda dapat menghapus pengguna dari ruang obrolan di aplikasi obrolan Anda.

Langkah Berikutnya

Sebagai percobaan, coba implementasikan tindakan lain di suatu ruang, seperti memutus koneksi pengguna lain.

IVSPesan Klien ObrolanSDK: Bereaksi & Bereaksi Praktik Terbaik Asli

Dokumen ini menjelaskan praktik paling penting dalam menggunakan Amazon IVS Chat Messaging SDK for React dan React Native. Informasi ini akan memungkinkan Anda untuk membangun

fungsionalitas obrolan khas di dalam aplikasi React, dan memberi Anda latar belakang yang Anda butuhkan untuk menyelam lebih dalam ke bagian yang lebih canggih dari Pesan IVS ObrolanSDK.

Membuat Hook ChatRoom Initializer

Kelas `ChatRoom` berisi metode dan pendengar obrolan inti untuk mengelola status koneksi serta mendengarkan peristiwa seperti pesan diterima dan pesan dihapus. Di sini, kami akan menunjukkan cara menyimpan instans obrolan dengan benar di hook.

Implementasi

TypeScript

```
// useChatRoom.ts

import React from 'react';
import { ChatRoom, ChatRoomConfig } from 'amazon-ivs-chat-messaging';

export const useChatRoom = (config: ChatRoomConfig) => {
  const [room] = React.useState(() => new ChatRoom(config));

  return { room };
};
```

JavaScript

```
import React from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';

export const useChatRoom = (config) => {
  const [room] = React.useState(() => new ChatRoom(config));

  return { room };
};
```

Catatan: Kami tidak menggunakan metode `dispatch` dari hook `setState` karena Anda tidak dapat memperbarui parameter konfigurasi dengan cepat. SDKMembuat instance sekali, dan tidak mungkin untuk memperbarui penyedia token.

Penting: Gunakan hook penginisialisasi `ChatRoom` sekali untuk menginisialisasi instans ruang obrolan baru.

Contoh

TypeScript/JavaScript:

```
// ...

const MyChatScreen = () => {
  const userId = 'Mike';
  const { room } = useChatRoom({
    regionOrUrl: SOCKET_URL,
    tokenProvider: () => tokenProvider(ROOM_ID, ['SEND_MESSAGE']),
  });

  const handleConnect = () => {
    room.connect();
  };

  // ...
};

// ...
```

Mendengarkan Status Koneksi

Secara opsional, Anda dapat berlangganan pembaruan status koneksi di hook ruang obrolan Anda.

Implementasi

TypeScript

```
// useChatRoom.ts

import React from 'react';
import { ChatRoom, ChatRoomConfig, ConnectionState } from 'amazon-ivs-chat-
messaging';

export const useChatRoom = (config: ChatRoomConfig) => {
  const [room] = useState(() => new ChatRoom(config));

  const [state, setState] = React.useState<ConnectionState>('disconnected');

  React.useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {
```

```
      setState('connecting');
    });

    const unsubscribeOnConnected = room.addListener('connect', () => {
      setState('connected');
    });

    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
      setState('disconnected');
    });

    return () => {
      unsubscribeOnConnecting();
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, []);

  return { room, state };
};
```

JavaScript

```
// useChatRoom.js

import React from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';

export const useChatRoom = (config) => {
  const [room] = useState(() => new ChatRoom(config));

  const [state, setState] = React.useState('disconnected');

  React.useEffect(() => {
    const unsubscribeOnConnecting = room.addListener('connecting', () => {
      setState('connecting');
    });

    const unsubscribeOnConnected = room.addListener('connect', () => {
      setState('connected');
    });

    const unsubscribeOnDisconnected = room.addListener('disconnect', () => {
```

```
      setState('disconnected');
    });

    return () => {
      unsubscribeOnConnecting();
      unsubscribeOnConnected();
      unsubscribeOnDisconnected();
    };
  }, []);

  return { room, state };
};
```

ChatRoom Penyedia Instance

Untuk menggunakan hook di komponen lain (guna menghindari penggunaan prop drilling), Anda dapat membuat penyedia ruang obrolan dengan menggunakan `context` React.

Implementasi

TypeScript

```
// ChatRoomContext.tsx

import React from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';

const ChatRoomContext = React.createContext<ChatRoom | undefined>(undefined);

export const useChatRoomContext = () => {
  const context = React.useContext(ChatRoomContext);

  if (context === undefined) {
    throw new Error('useChatRoomContext must be within ChatRoomProvider');
  }

  return context;
};

export const ChatRoomProvider = ChatRoomContext.Provider;
```

JavaScript

```
// ChatRoomContext.jsx

import React from 'react';
import { ChatRoom } from 'amazon-ivs-chat-messaging';

const ChatRoomContext = React.createContext(undefined);

export const useChatRoomContext = () => {
  const context = React.useContext(ChatRoomContext);

  if (context === undefined) {
    throw new Error('useChatRoomContext must be within ChatRoomProvider');
  }

  return context;
};

export const ChatRoomProvider = ChatRoomContext.Provider;
```

Contoh

Setelah membuat `ChatRoomProvider`, Anda dapat menggunakan instans Anda dengan `useChatRoomContext`.

Penting: Letakkan penyedia di tingkat root hanya jika Anda memerlukan akses ke `context` di antara layar obrolan dan komponen lain di tengah, untuk menghindari render ulang yang tidak perlu jika Anda mendengarkan koneksi. Jika tidak, letakkan penyedia sedekat mungkin ke layar obrolan.

TypeScript/JavaScript:

```
// AppContainer

const AppContainer = () => {
  const { room } = useChatRoom({
    regionOrUrl: SOCKET_URL,
    tokenProvider: () => tokenProvider(ROOM_ID, ['SEND_MESSAGE']),
  });

  return (
    <ChatRoomProvider value={room}>
```

```
      <MyChatScreen />
    </ChatRoomProvider>
  );
};

// MyChatScreen

const MyChatScreen = () => {
  const room = useChatRoomContext();

  const handleConnect = () => {
    room.connect();
  };
  // ...
};

// ...
```

Membuat Pendengar Pesan

Agar tetap mengetahui semua pesan masuk, Anda harus berlangganan peristiwa `message` dan `deleteMessage`. Berikut adalah beberapa kode yang menyediakan pesan obrolan untuk komponen Anda.

Penting: Untuk tujuan performa, kami memisahkan `ChatMessageContext` dari `ChatRoomProvider`, karena kami mungkin mendapatkan banyak render ulang saat pendengar pesan-obrolan memperbarui status pesannya. Ingatlah untuk menerapkan `ChatMessageContext` pada komponen tempat Anda akan menggunakan `ChatMessageProvider`.

Implementasi

TypeScript

```
// ChatMessagesContext.tsx

import React from 'react';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useChatRoomContext } from './ChatRoomContext';

const ChatMessagesContext = React.createContext<ChatMessage[] |
 undefined>(undefined);
```

```
export const useChatMessagesContext = () => {
  const context = React.useContext(ChatMessagesContext);

  if (context === undefined) {
    throw new Error('useChatMessagesContext must be within ChatMessagesProvider);
  }

  return context;
};

export const ChatMessagesProvider = ({ children }: { children: React.ReactNode }) =>
 {
  const room = useChatRoomContext();

  const [messages, setMessages] = React.useState<ChatMessage[]>([]);

  React.useEffect(() => {
    const unsubscribeOnMessageReceived = room.addListener('message', (message) => {
      setMessages((msgs) => [message, ...msgs]);
    });

    const unsubscribeOnMessageDeleted = room.addListener('messageDelete',
 (deleteEvent) => {
      setMessages((prev) => prev.filter((message) => message.id !==
 deleteEvent.messageId));
    });

    return () => {
      unsubscribeOnMessageDeleted();
      unsubscribeOnMessageReceived();
    };
  }, [room]);

  return <ChatMessagesContext.Provider value={messages}>{children}</
ChatMessagesContext.Provider>;
};
```

JavaScript

```
// ChatMessagesContext.jsx

import React from 'react';
import { useChatRoomContext } from './ChatRoomContext';
```

```
const ChatMessagesContext = React.createContext(undefined);

export const useChatMessagesContext = () => {
  const context = React.useContext(ChatMessagesContext);

  if (context === undefined) {
    throw new Error('useChatMessagesContext must be within ChatMessagesProvider);
  }

  return context;
};

export const ChatMessagesProvider = ({ children }) => {
  const room = useChatRoomContext();

  const [messages, setMessages] = React.useState([]);

  React.useEffect(() => {
    const unsubscribeOnMessageReceived = room.addListener('message', (message) => {
      setMessages((msgs) => [message, ...msgs]);
    });

    const unsubscribeOnMessageDeleted = room.addListener('messageDelete',
 (deleteEvent) => {
      setMessages((prev) => prev.filter((message) => message.id !==
 deleteEvent.messageId));
    });

    return () => {
      unsubscribeOnMessageDeleted();
      unsubscribeOnMessageReceived();
    };
  }, [room]);

  return <ChatMessagesContext.Provider value={messages}>{children}</
ChatMessagesContext.Provider>;
};
```

Contoh di React

Penting: Ingatlah untuk membungkus kontainer pesan Anda dengan `ChatMessagesProvider`. Baris `Message` adalah contoh komponen yang menampilkan konten pesan.

TypeScript/JavaScript:

```
// your message list component...

import React from 'react';
import { useChatMessagesContext } from './ChatMessagesContext';

const MessageListContainer = () => {
  const messages = useChatMessagesContext();

  return (
    <React.Fragment>
      {messages.map((message) => (
        <MessageRow message={message} />
      ))}
    </React.Fragment>
  );
};
```

Contoh di React Native

Secara default, `ChatMessage` berisi `id`, yang digunakan secara otomatis sebagai kunci React di `FlatList` untuk setiap baris; oleh karena itu, Anda tidak perlu menyediakan `keyExtractor`.

TypeScript

```
// MessageListContainer.tsx

import React from 'react';
import { ListRenderItemInfo, FlatList } from 'react-native';
import { ChatMessage } from 'amazon-ivs-chat-messaging';
import { useChatMessagesContext } from './ChatMessagesContext';

const MessageListContainer = () => {
  const messages = useChatMessagesContext();

  const renderItem = useCallback(({ item }: ListRenderItemInfo<ChatMessage>) =>
 <MessageRow />, []);
```

```
  return <FlatList data={messages} renderItem={renderItem} />;
};
```

JavaScript

```
// MessageListContainer.jsx

import React from 'react';
import { FlatList } from 'react-native';
import { useChatMessagesContext } from './ChatMessagesContext';

const MessageListContainer = () => {
  const messages = useChatMessagesContext();

  const renderItem = useCallback(({ item }) => <MessageRow />, []);

  return <FlatList data={messages} renderItem={renderItem} />;
};
```

Beberapa Instans Ruang Obrolan dalam Aplikasi

Jika Anda menggunakan beberapa ruang obrolan secara bersamaan di aplikasi Anda, kami mengusulkan untuk membuat setiap penyedia bagi setiap obrolan dan menggunakannya di penyedia obrolan. Di dalam contoh ini, kami sedang membuat obrolan Bot Bantuan dan Bantuan Pelanggan. Kami membuat penyedia untuk keduanya.

TypeScript

```
// SupportChatProvider.tsx

import React from 'react';
import { SUPPORT_ROOM_ID, SOCKET_URL } from '../../config';
import { tokenProvider } from '../tokenProvider';
import { ChatRoomProvider } from './ChatRoomContext';
import { useChatRoom } from './useChatRoom';

export const SupportChatProvider = ({ children }: { children: React.ReactNode }) =>
 {
  const { room } = useChatRoom({
    regionOrUrl: SOCKET_URL,
```

```
    tokenProvider: () => tokenProvider(SUPPORT_ROOM_ID, ['SEND_MESSAGE']),
  });

  return <ChatRoomProvider value={room}>{children}</ChatRoomProvider>;
};

// SalesChatProvider.tsx

import React from 'react';
import { SALES_ROOM_ID, SOCKET_URL } from '../../config';
import { tokenProvider } from '../tokenProvider';
import { ChatRoomProvider } from './ChatRoomContext';
import { useChatRoom } from './useChatRoom';

export const SalesChatProvider = ({ children }: { children: React.ReactNode }) => {
  const { room } = useChatRoom({
    regionOrUrl: SOCKET_URL,
    tokenProvider: () => tokenProvider(SALES_ROOM_ID, ['SEND_MESSAGE']),
  });

  return <ChatRoomProvider value={room}>{children}</ChatRoomProvider>;
};
```

JavaScript

```
// SupportChatProvider.jsx

import React from 'react';
import { SUPPORT_ROOM_ID, SOCKET_URL } from '../../config';
import { tokenProvider } from '../tokenProvider';
import { ChatRoomProvider } from './ChatRoomContext';
import { useChatRoom } from './useChatRoom';

export const SupportChatProvider = ({ children }) => {
  const { room } = useChatRoom({
    regionOrUrl: SOCKET_URL,
    tokenProvider: () => tokenProvider(SUPPORT_ROOM_ID, ['SEND_MESSAGE']),
  });

  return <ChatRoomProvider value={room}>{children}</ChatRoomProvider>;
};

// SalesChatProvider.jsx
```

```
import React from 'react';
import { SALES_ROOM_ID, SOCKET_URL } from '../../config';
import { tokenProvider } from '../tokenProvider';
import { ChatRoomProvider } from './ChatRoomContext';
import { useChatRoom } from './useChatRoom';

export const SalesChatProvider = ({ children }) => {
  const { room } = useChatRoom({
    regionOrUrl: SOCKET_URL,
    tokenProvider: () => tokenProvider(SALES_ROOM_ID, ['SEND_MESSAGE']),
  });

  return <ChatRoomProvider value={room}>{children}</ChatRoomProvider>;
};
```

Contoh di React

Sekarang Anda dapat memanfaatkan penyedia obrolan berbeda yang menggunakan `ChatRoomProvider` yang sama. Kemudian, Anda dapat menggunakan kembali `useChatRoomContext` yang sama di dalam setiap layar/tampilan.

TypeScript/JavaScript:

```
// App.tsx / App.jsx

const App = () => {
  return (
    <Routes>
      <Route
        element={
          <SupportChatProvider>
            <SupportChatScreen />
          </SupportChatProvider>
        }
      />
      <Route
        element={
          <SalesChatProvider>
            <SalesChatScreen />
          </SalesChatProvider>
        }
```

```
      />
    </Routes>
  );
};
```

Contoh di React Native

TypeScript/JavaScript:

```
// App.tsx / App.jsx

const App = () => {
  return (
    <Stack.Navigator>
      <Stack.Screen name="SupportChat">
        <SupportChatProvider>
          <SupportChatScreen />
        </SupportChatProvider>
      </Stack.Screen>
      <Stack.Screen name="SalesChat">
        <SalesChatProvider>
          <SalesChatScreen />
        </SalesChatProvider>
      </Stack.Screen>
    </Stack.Navigator>
  );
};
```

TypeScript/JavaScript:

```
// SupportChatScreen.tsx / SupportChatScreen.jsx

// ...

const SupportChatScreen = () => {
  const room = useChatRoomContext();

  const handleConnect = () => {
    room.connect();
  };

  return (
    <>
```

```
      <Button title="Connect" onPress={handleConnect} />
      <MessageListContainer />
    </>
  );
};

// SalesChatScreen.tsx / SalesChatScreen.jsx

// ...

const SalesChatScreen = () => {
  const room = useChatRoomContext();

  const handleConnect = () => {
    room.connect();
  };

  return (
    <>
      <Button title="Connect" onPress={handleConnect} />
      <MessageListContainer />
    </>
  );
};
```

Keamanan IVS Obrolan Amazon

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan cloud dan keamanan dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara berkala menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program kepatuhan AWS.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan IVS Obrolan Amazon. Topik berikut menunjukkan cara mengonfigurasi IVS Obrolan Amazon untuk memenuhi tujuan keamanan dan kepatuhan Anda.

Topik

IVSPerlindungan Data Obrolan

Untuk data yang dikirim ke Amazon Interactive Video Service (IVS) Chat, perlindungan data berikut tersedia:

- Lalu lintas IVS Obrolan Amazon digunakan WSS untuk menjaga keamanan data saat transit.
- Token IVS Obrolan Amazon dienkripsi menggunakan kunci yang dikelola KMS pelanggan.

Amazon IVS Chat tidak mengharuskan Anda memberikan data pelanggan (pengguna akhir) apa pun. Tidak ada bidang di ruang obrolan, input, atau grup keamanan input yang meminta Anda agar memberikan data pelanggan (pengguna akhir).

Jangan masukkan informasi pengidentifikasi yang sensitif, seperti nomor rekening pelanggan (pengguna akhir) Anda, ke dalam bidang isian bentuk bebas seperti bidang Nama. Ini termasuk ketika Anda bekerja dengan IVS konsol Amazon atauAPI, AWS CLI, atau AWS SDKs. Setiap bagian data yang Anda masukkan ke Amazon IVS Chat mungkin disertakan dalam log diagnostik.

Aliran tidak end-to-end dienkripsi; aliran dapat ditransmisikan secara tidak terenkripsi secara internal dalam jaringan, untuk diproses. IVS

Identity and Access Management di IVS Chat

AWS Identity and Access Management (IAM) adalah AWS layanan yang membantu administrator akun mengontrol akses ke AWS sumber daya dengan aman. Lihat Identity and Access Management IVS di Panduan Pengguna Streaming IVS Latensi Rendah.

Audiens

Cara Anda menggunakan IAM berbeda, tergantung pada pekerjaan yang Anda lakukan di AmazonIVS. Lihat Audiens di Panduan Pengguna Streaming IVS Latensi Rendah.

Bagaimana Amazon IVS Bekerja dengan IAM

Sebelum Anda dapat membuat IVS API permintaan Amazon, Anda harus membuat satu atau beberapa IAM identitas (pengguna, grup, dan peran) dan IAM kebijakan, lalu lampirkan kebijakan ke identitas. Diperlukan waktu hingga beberapa menit agar izin disebarkan; sampai saat itu, API permintaan ditolak.

Untuk tampilan tingkat tinggi tentang cara IVS kerja AmazonIAM, lihat AWS Layanan yang Bekerja dengan IAM di Panduan IAM Pengguna.

Identitas

Anda dapat membuat IAM identitas untuk memberikan otentikasi bagi orang dan proses di akun Anda AWS . IAMgrup adalah kumpulan IAM pengguna yang dapat Anda kelola sebagai satu unit. Lihat Identitas (Pengguna, Grup, dan Peran) di Panduan IAM Pengguna.

Kebijakan

Kebijakan adalah dokumen JSON kebijakan izin yang terdiri dari elemen. Lihat Kebijakan di Panduan Pengguna Streaming IVS Latensi Rendah.

Amazon IVS Chat mendukung tiga elemen:

- Tindakan — Tindakan kebijakan untuk IVS Obrolan Amazon menggunakan `ivschat` awalan sebelum tindakan. Misalnya, untuk memberikan izin kepada seseorang untuk membuat ruang IVS Obrolan Amazon dengan `CreateRoom` API metode IVS Obrolan Amazon, Anda menyertakan `ivschat:CreateRoom` tindakan tersebut dalam kebijakan untuk orang tersebut. Pernyataan kebijakan harus memuat elemen `Action` atau `NotAction`.
- Sumber daya - Sumber daya ruang IVS Obrolan Amazon memiliki ARNformat berikut:

  ```
  arn:aws:ivschat:${Region}:${Account}:room/${roomId}
  ```

  Misalnya, untuk menentukan `VgNkEJgOVX9N` ruangan dalam pernyataan Anda, gunakan iniARN:

  ```
  "Resource": "arn:aws:ivschat:us-west-2:123456789012:room/VgNkEJgOVX9N"
  ```

  Beberapa tindakan IVS Obrolan Amazon, seperti untuk membuat sumber daya, tidak dapat dilakukan pada sumber daya tertentu. Dalam kasus tersebut, Anda harus menggunakan wildcard (*):

  ```
  "Resource":"*"
  ```

- Ketentuan - IVS Obrolan Amazon mendukung beberapa kunci kondisi global:`aws:RequestTag`,`aws:TagKeys`, dan`aws:ResourceTag`.

Anda dapat memanfaatkan variabel sebagai placeholder dalam sebuah kebijakan. Misalnya, Anda dapat memberikan izin IAM pengguna untuk mengakses sumber daya hanya jika ditandai dengan nama pengguna IAM pengguna. Lihat Variabel dan Tag di Panduan IAM Pengguna.

Amazon IVS menyediakan kebijakan AWS terkelola yang dapat digunakan untuk memberikan sekumpulan izin yang telah dikonfigurasi sebelumnya ke identitas (hanya baca atau akses penuh). Anda dapat memilih untuk menggunakan kebijakan terkelola alih-alih kebijakan berbasis identitas yang ditunjukkan di bawah ini. Untuk detailnya, lihat Kebijakan Terkelola untuk IVS Obrolan Amazon.

Otorisasi Berdasarkan Tag Amazon IVS

Anda dapat melampirkan tag ke sumber daya IVS Obrolan Amazon atau meneruskan tag dalam permintaan ke IVS Obrolan Amazon. Untuk mengontrol akses berdasarkan tanda, Anda harus memberikan informasi tanda di elemen persyaratan sebuah kebijakan dengan menggunakan kunci syarat `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`. Untuk informasi selengkapnya tentang menandai sumber daya IVS Obrolan Amazon, lihat “Menandai” di Referensi IVSObrolan API.

Peran

Lihat IAMPeran dan Kredenal Keamanan Sementara di IAMPanduan Pengguna.

IAMPeran adalah entitas dalam AWS akun Anda yang memiliki izin khusus.

Amazon IVS mendukung penggunaan kredensil keamanan sementara. Anda dapat menggunakan kredensi sementara untuk masuk dengan federasi, mengambil IAM peran, atau mengambil peran lintas akun. Anda memperoleh kredensi keamanan sementara dengan memanggil API operasi Layanan Token AWS Keamanan seperti `AssumeRole` atau. `GetFederationToken`

Hak Akses Istimewa dan Tidak Istimewa

APIsumber daya memiliki akses istimewa. Akses pemutaran yang tidak memiliki hak istimewa dapat diatur melalui saluran pribadi; lihat Menyiapkan Saluran IVS Pribadi.

Praktik Terbaik untuk Kebijakan

Lihat Praktik IAM Terbaik di Panduan IAM Pengguna.

Kebijakan berbasis identitas adalah pilihan yang sangat tepat. Mereka menentukan apakah seseorang dapat membuat, mengakses, atau menghapus IVS sumber daya Amazon di akun Anda. Tindakan ini dapat menimbulkan biaya untuk AWS akun Anda. Ikuti rekomendasi ini:

- Berikan hak akses paling rendah — Ketika Anda membuat kebijakan kustom, berikan hanya izin yang diperlukan untuk melakukan tugas. Mulai dengan set izin minimum dan berikan izin tambahan sesuai kebutuhan. Hal itu lebih aman daripada memulai dengan izin yang terlalu fleksibel, lalu mencoba memperketatnya nanti. Secara khusus, simpan `ivschat:*` untuk akses admin; jangan menggunakannya dalam aplikasi.
- Aktifkan autentikasi multi-faktor (MFA) untuk operasi sensitif — Untuk keamanan ekstra, IAM pengguna harus menggunakannya MFA untuk mengakses sumber daya atau API operasi yang sensitif.
- Gunakan syarat kebijakan untuk keamanan ekstra — Selama bisa dilakukan, tentukan persyaratan agar kebijakan berbasis identitas Anda mengizinkan akses ke sumber daya. Misalnya, Anda dapat menulis persyaratan untuk menentukan rentang alamat IP yang diizinkan untuk mengajukan permintaan. Anda juga dapat menulis kondisi untuk mengizinkan permintaan hanya dalam tanggal atau rentang waktu tertentu, atau untuk meminta penggunaan SSL atauMFA.

Contoh Kebijakan Berbasis Identitas

Gunakan IVS Konsol Amazon

Untuk mengakses IVS konsol Amazon, Anda harus memiliki seperangkat izin minimum yang memungkinkan Anda membuat daftar dan melihat detail tentang sumber daya IVS Obrolan Amazon di AWS akun Anda. Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tersebut tidak akan berfungsi sebagaimana mestinya untuk identitas dengan kebijakan tersebut. Untuk memastikan akses ke IVS konsol Amazon, lampirkan kebijakan berikut ke identitas (lihat Menambahkan dan Menghapus IAM Izin di Panduan IAM Pengguna).

Bagian-bagian dari kebijakan berikut menyediakan akses ke:

- Semua titik API akhir IVS Obrolan Amazon
- Kuota layanan IVS Obrolan Amazon Anda
- Mencantumkan lambda dan menambahkan izin untuk lambda yang dipilih untuk moderasi Obrolan Amazon IVS
- Amazon Cloudwatch untuk mendapatkan metrik bagi sesi obrolan Anda

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Action": "ivschat:*",
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "servicequotas:ListServiceQuotas"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "cloudwatch:GetMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "lambda:AddPermission",
        "lambda:ListFunctions"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Kebijakan Berbasis Sumber Daya untuk Obrolan Amazon IVS

Anda harus memberikan izin layanan IVS Obrolan Amazon untuk memanggil sumber daya lambda Anda untuk meninjau pesan. Untuk melakukannya, ikuti petunjuk di Menggunakan kebijakan berbasis sumber daya untuk Lambda (dalam Panduan Pengembang AWS Lambda) dan isi AWS kolom seperti yang ditentukan di bawah ini.

Untuk mengontrol akses ke sumber daya lambda, Anda dapat menggunakan persyaratan berdasarkan:

- `SourceArn` — Contoh kebijakan kami memanfaatkan wildcard (*) agar semua ruang di akun Anda dapat menginvokasi lambda. Secara opsional, Anda dapat menentukan ruang di akun untuk mengizinkan hanya ruang tersebut yang dapat menginvokasi lambda.
- `SourceAccount`— Dalam contoh kebijakan di bawah ini, ID AWS akun adalah`123456789012`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Principal": {
            "Service": "ivschat.amazonaws.com"
         },
         "Action": [
            "lambda:InvokeFunction"
         ],
         "Effect": "Allow",
         "Resource": "arn:aws:lambda:us-west-2:123456789012:function:name",
         "Condition": {
            "StringEquals": {
               "AWS:SourceAccount": "123456789012"
            },
            "ArnLike": {
               "AWS:SourceArn": "arn:aws:ivschat:us-west-2:123456789012:room/*"
            }
         }
      }
   ]
}
```

Pemecahan Masalah

Lihat Pemecahan Masalah di Panduan Pengguna Streaming IVS Latensi Rendah untuk informasi tentang mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Obrolan Amazon dan. IVS IAM

Kebijakan Terkelola untuk IVS Obrolan

Kebijakan AWS terkelola adalah kebijakan mandiri yang dibuat dan dikelola olehAWS. Lihat Kebijakan Terkelola untuk Amazon IVS di Panduan Pengguna Streaming IVS Latensi Rendah.

Menggunakan Peran Tertaut Layanan untuk Obrolan IVS

Amazon IVS menggunakan peran AWS IAM terkait layanan. Lihat Menggunakan Peran Tertaut Layanan untuk Amazon IVS di Panduan Pengguna Streaming IVSLatensi Rendah.

IVSPencatatan dan Pemantauan Obrolan

Untuk mencatat kinerja dan/atau operasi, gunakan Amazon CloudTrail. Lihat Mencatat IVS API Panggilan Amazon dengan AWS CloudTrail di Panduan Pengguna Streaming IVS Latensi Rendah.

IVSTanggapan Insiden Obrolan

Untuk mendeteksi atau memperingatkan insiden, Anda dapat memantau kesehatan streaming Anda melalui EventBridge acara Amazon. Lihat Menggunakan Amazon EventBridge dengan AmazonIVS: untuk Streaming Latensi Rendah dan Streaming Waktu Nyata.

Gunakan Dasbor AWS Kesehatan untuk informasi tentang kesehatan Amazon secara keseluruhan IVS (berdasarkan wilayah).

IVSKetahanan Obrolan

IVSAPIsmenggunakan infrastruktur AWS global dan dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. Lihat IVSKetahanan dalam Panduan Pengguna Streaming IVSLatensi Rendah.

IVSKeamanan Infrastruktur Obrolan

Sebagai layanan terkelola, Amazon IVS dilindungi oleh prosedur keamanan jaringan AWS global. Hal ini dijelaskan dalam Praktik Terbaik untuk Keamanan, Identitas, dan Kepatuhan.

APIPanggilan

Anda menggunakan API panggilan yang AWS dipublikasikan untuk mengakses Amazon IVS melalui jaringan. Lihat APIPanggilan di bawah Keamanan Infrastruktur di Panduan Pengguna Streaming IVS Latensi Rendah.

IVSObrolan Amazon

Penyerapan dan pengiriman pesan IVS Obrolan Amazon terjadi melalui WSS koneksi terenkripsi ke edge kami. Amazon IVS Messaging API menggunakan koneksi terenkripsi. HTTPS Seperti halnya

streaming dan pemutaran video, TLS versi 1.2 atau yang lebih baru diperlukan dan data pesan dapat dikirimkan tanpa enkripsi secara internal untuk diproses.

IVSService Quotas Obrolan

Berikut ini adalah kuota dan batasan layanan untuk titik akhir obrolan Amazon Interactive Video Service (IVS), sumber daya, dan operasi lainnya. Kuota layanan (juga dikenal sebagai batas) adalah jumlah maksimum sumber daya layanan atau operasi untuk AWS akun Anda. Artinya, batas ini per AWS akun, kecuali disebutkan lain dalam tabel. Lihat juga AWSService Quotas.

Anda menggunakan titik akhir untuk terhubung secara terprogram ke layanan. AWS Lihat juga Titik Akhir AWS Layanan.

Semua kuota diberlakukan per wilayah.

Peningkatan Kuota Layanan

Untuk kuota yang dapat disesuaikan, Anda dapat meminta kenaikan tarif melalui AWSkonsol. Gunakan konsol juga untuk melihat informasi tentang kuota layanan.

APIkuota tingkat panggilan tidak dapat disesuaikan.

APIKuota Tarif Panggilan

Tipe Titik Akhir	Titik Akhir	Default
Perpesanan	`DeleteMessage`	100 TPS
Perpesanan	`DisconnectUser`	100 TPS
Perpesanan	`SendEvent`	100 TPS
Token obrolan	`CreateChatToken`	200 TPS
Konfigurasi Pembuatan Log	`CreateLoggingConfiguration`	3 TPS
Konfigurasi Pembuatan Log	`DeleteLoggingConfiguration`	3 TPS
Konfigurasi Pembuatan Log	`GetLoggingConfiguration`	3 TPS
Konfigurasi Pembuatan Log	`ListLoggingConfigurations`	3 TPS

Tipe Titik Akhir	Titik Akhir	Default
Konfigurasi Pembuatan Log	`UpdateLoggingConfiguration`	3 TPS
Ruang	`CreateRoom`	5 TPS
Ruang	`DeleteRoom`	5 TPS
Ruang	`GetRoom`	5 TPS
Ruang	`ListRooms`	5 TPS
Ruang	`UpdateRoom`	5 TPS
Tanda	`ListTagsForResource`	10 TPS
Tanda	`TagResource`	10 TPS
Tanda	`UntagResource`	10 TPS

Kuota Lainnya

Sumber Daya atau Fitur	Default	Dapat disesuaikan	Deskripsi
Koneksi obrolan bersamaan	50.000	Ya	Jumlah maksimum koneksi obrolan bersamaan per akun, di semua ruang Anda dalam satu Wilayah AWS.
Konfigurasi pembuatan log	10	Ya	Jumlah maksimum konfigura si pembuatan log yang dapat dibuat per akun di Wilayah AWS saat ini.
Periode batas waktu handler tinjauan pesan	200	Tidak	Periode batas waktu dalam milidetik untuk semua handler tinjauan pesan Anda

Sumber Daya atau Fitur	Default	Dapat disesuaikan	Deskripsi
			di Wilayah AWS saat ini. Jika periode ini terlampaui, pesan diizinkan atau ditolak bergantung pada nilai bidang `fallbackResult` yang Anda konfigurasikan untuk handler tinjauan pesan.
Tarif DeleteMessage permintaan di semua kamar Anda	100	Ya	Jumlah maksimum DeleteMessage permintaan yang dapat dilakukan per detik di semua kamar Anda. Permintaan dapat berasal dari IVS Obrolan Amazon API atau Pesan IVS Obrolan Amazon API (WebSocket).
Tarif DisconnectUser permintaan di semua kamar Anda	100	Ya	Jumlah maksimum DisconnectUser permintaan yang dapat dilakukan per detik di semua kamar Anda. Permintaan dapat berasal dari IVS Obrolan Amazon API atau Pesan IVS Obrolan Amazon API (WebSocket).
Tingkat permintaan perpesanan per koneksi	10	Tidak	Jumlah maksimum permintaan perpesanan per detik yang dapat dilakukan oleh koneksi obrolan.

Sumber Daya atau Fitur	Default	Dapat disesuaikan	Deskripsi
Tarif SendMessage permintaan di semua kamar Anda	1000	Ya	Jumlah maksimum SendMessage permintaan yang dapat dilakukan per detik di semua kamar Anda. Permintaan ini berasal dari Amazon IVS Chat Messaging API (WebSocket).
Tarif SendMessage permintaan per kamar	100	Tidak (tetapi dapat dikonfigurasi melalui) API	Jumlah maksimum SendMessage permintaan yang dapat dilakukan per detik untuk salah satu kamar Anda. Ini dapat dikonfigurasi dengan `maximumMessageRatePerSecond` bidang CreateRoomdan UpdateRoom. Permintaan ini berasal dari Amazon IVS Chat Messaging API (WebSocket).
Ruang	50.000	Ya	Jumlah maksimum ruang obrolan per akun, per Wilayah AWS.

Integrasi Service Quotas dengan Metrik Penggunaan CloudWatch

Anda dapat menggunakan CloudWatch untuk secara proaktif mengelola kuota layanan Anda, melalui metrik CloudWatch penggunaan. Anda dapat menggunakan metrik ini untuk memvisualisasikan penggunaan layanan Anda saat ini pada CloudWatch grafik dan dasbor. Metrik penggunaan IVS Obrolan Amazon sesuai dengan kuota layanan IVS Obrolan Amazon.

Anda dapat menggunakan fungsi matematika CloudWatch metrik untuk menampilkan kuota layanan untuk sumber daya tersebut pada grafik Anda. Anda juga dapat mengonfigurasi alarm yang memberi tahu Anda ketika penggunaan mendekati kuota layanan.

Untuk mengakses metrik penggunaan:

1. Buka konsol Service Quotas di https://console.aws.amazon.com/servicequotas/
2. Di panel navigasi, pilih AWSlayanan.
3. Dari daftar AWS layanan, cari dan pilih Obrolan Layanan Video Interaktif Amazon.
4. Dalam daftar Kuota layanan, pilih kuota layanan yang dicari. Sebuah halaman baru akan terbuka dengan informasi tentang kuota layanan/metrik.

Sebagai alternatif, Anda bisa mendapatkan metrik ini melalui konsol. CloudWatch Di bawah AWSNamespaces, pilih Usage. Kemudian, dari daftar Layanan, pilih IVSObrolan. (Lihat Memantau IVS Obrolan Amazon.)

Di namespace AWS/Usage, Amazon IVS Chat menyediakan metrik berikut:

Nama Metrik	Deskripsi
`ResourceCount`	Jumlah sumber daya yang telah ditentukan yang sedang berjalan di akun Anda. Sumber daya tersebut ditentukan oleh dimensi yang dikaitkan dengan metrik. Statistik valid: Maksimum (jumlah maksimum sumber daya yang digunakan selama periode 1 menit).

Dimensi berikut digunakan untuk memperbaiki metrik penggunaan:

Dimensi	Deskripsi
Layanan	Nama AWS layanan yang berisi sumber daya. Nilai valid: `IVS Chat`.
Kelas	Kelas sumber daya yang akan dilacak. Nilai valid: `None`.
Tipe	Tipe sumber daya yang sedang ditelusuri. Nilai valid: `Resource`.
Sumber Daya	Nama sumber daya AWS. Nilai valid: `ConcurrentChatConnections` .

Dimensi	Deskripsi
	Metrik ConcurrentChatConnections penggunaan adalah salinan dari yang ada di IVSChat namespace AWS/(dengan dimensi None), seperti yang dijelaskan dalam Memantau Obrolan Amazon IVS.

Membuat CloudWatch Alarm untuk Metrik Penggunaan

Untuk membuat CloudWatch alarm berdasarkan metrik penggunaan IVS Obrolan Amazon:

1. Dari konsol Service Quotas, pilih kuota layanan yang dicari, sebagaimana dijelaskan di atas. Saat ini, alarm hanya dapat dibuat untuk ConcurrentChatConnections.
2. Di bagian CloudWatch alarm Amazon, pilih Buat.
3. Dari daftar menu tarik-turun Ambang batas alarm, pilih persentase nilai kuota Anda yang diterapkan, yang ingin Anda tetapkan sebagai nilai alarm.
4. Untuk Nama alarm, masukkan nama untuk alarm.
5. Pilih Buat.

Memecahkan Masalah IVS Obrolan

Dokumen ini menjelaskan praktik terbaik dan tips pemecahan masalah untuk Amazon Interactive Video Service (IVS) Chat. Perilaku yang terkait dengan IVS Obrolan sering kali berbeda dari perilaku yang terkait dengan IVS video. Untuk informasi selengkapnya, lihat Memulai IVS Obrolan Amazon.

Topik:

Mengapa koneksi IVS obrolan tidak terputus saat ruangan dihapus?

Ketika sumber daya ruang obrolan dihapus, jika ruang sedang digunakan secara aktif, klien obrolan yang terhubung ke ruang tersebut tidak terputus secara otomatis. Koneksi hilang jika/ketika aplikasi obrolan menyegarkan token obrolan. Atau, pemutusan semua pengguna secara manual harus dilakukan untuk menghapus semua pengguna dari ruang obrolan.

Daftar istilah IVS

Lihat juga AWSglosariumnya. Pada tabel di bawah ini, LL adalah singkatan dari streaming IVS latensi rendah; RT, IVS streaming waktu nyata.

Istilah	Deskripsi	LL	RT	Obrolan
AAC	Pengodean Audio Tingkat Lanjut. AACadalah standar pengkodean audio untuk kompresi audio digital lossy. Dirancang untuk menjadi penerus MP3 format, AAC umumnya mencapai kualitas suara yang lebih tinggi daripada MP3 pada bitrate yang sama. AACtelah distandarisasi oleh ISO dan IEC sebagai bagian dari spesifikasi MPEG -2 dan MPEG -4.	✓	✓	
Streaming laju bit adaptif	Adaptive Bitrate (ABR) streaming memungkinkan IVS pemain untuk beralih ke bitrate yang lebih rendah ketika kualitas koneksi terganggu, dan untuk beralih kembali ke bitrate yang lebih tinggi ketika kualitas koneksi meningkat.	✓		
Streaming adaptif	Lihat Enkode berlapis dengan simulcast.		✓	
Pengguna administratif	AWSPengguna dengan akses administratif ke sumber daya dan layanan yang tersedia di AWS akun. Lihat Terminologi di Panduan Pengguna AWS Pengaturan.	✓	✓	✓
ARN	Nama Sumber Daya Amazon, pengenal unik untuk AWS sumber daya. ARNFormat tertentu tergantung pada jenis sumber daya. Untuk ARN format yang digunakan oleh IVS sumber daya, lihat di Referensi Otorisasi Layanan.	✓	✓	✓

Istilah	Deskripsi	LL	RT	Obrolan
Rasio aspek	Mendeskripsikan rasio lebar bingkai dengan tinggi bingkai. Misalnya, 16:9 adalah rasio aspek yang sesuai dengan resolusi Full HD atau 1080p.	✓	✓	
Mode audio	Suatu konfigurasi audio preset atau kustom yang dioptimalkan untuk berbagai tipe pengguna perangkat seluler dan perangkat yang mereka gunakan. Lihat IVSSiaranSDK: Mode Audio Seluler (Streaming Waktu Nyata).		✓	
AVC, H.264, -4 Bagian 10 MPEG	Advanced Video Coding, juga disebut sebagai H.264 atau MPEG -4 Bagian 10, standar kompresi video untuk kompresi video digital lossy.	✓	✓	
Penggantian latar belakang	Tipe filter kamera yang memungkinkan kreator streaming langsung untuk mengubah latar belakang mereka. Lihat Penggantian Latar Belakang di IVSSiaranSDK: Filter Kamera Pihak Ketiga (Streaming Waktu Nyata).		✓	
Laju bit	Suatu metrik streaming untuk jumlah bit yang ditransmisikan atau diterima per detik.	✓	✓	
Siaran, penyiar	Istilah lain untuk streaming, streamer.	✓		
Buffering	Suatu kondisi yang terjadi ketika perangkat pemutaran tidak dapat mengunduh konten sebelum konten seharusnya diputar. Buffering dapat bermanifestasi dalam beberapa cara: konten dapat berhenti dan mulai secara acak (juga dikenal sebagai gagap), konten dapat berhenti untuk jangka waktu yang lama (juga dikenal sebagai pembekuan), atau IVS pemain dapat menjeda pemutaran.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
Daftar putar rentang byte	Daftar putar yang lebih terperinci daripada daftar putar standarHLS. HLSDaftar putar standar terdiri dari file media 10 detik. Dengan daftar putar rentang byte, durasi segmen sama dengan interval keyframe yang dikonfigurasi untuk streaming. Daftar putar rentang byte hanya tersedia untuk siaran yang direkam secara otomatis ke bucket S3. Itu dibuat selain HLSdaftar putar. Lihat Daftar Putar Rentang Byte di Rekam Otomatis ke Amazon S3 (Streaming Latensi Rendah).	✓		
CBR	Laju Bit Konstan, sebuah metode kontrol laju untuk enkoder yang mempertahankan laju bit agar konsisten di seluruh pemutaran video, terlepas dari kejadian selama siaran. Jeda dalam tindakan mungkin ditambahkan untuk mencapai laju bit yang diinginkan, dan puncak mungkin dikuantisasi dengan menyesuaikan kualitas enkode agar cocok dengan laju bit target. Kami sangat menyarankan menggunakan CBR alih-alih VBR.	✓	✓	
CDN	Jaringan Pengiriman Konten atau Jaringan Distribusi Konten, sebuah solusi yang didistrib usikan secara geografis yang mengoptimalkan pengiriman konten seperti streaming video dengan membawanya lebih dekat dengan tempat pengguna berada.	✓		

Istilah	Deskripsi	LL	RT	Obrolan
Channel	IVSSumber daya yang menyimpan konfigurasi untuk streaming, termasuk server ingest, tombol streaming, pemutaran URL, dan opsi perekaman . Streamer menggunakan kunci streaming yang terkait dengan saluran untuk memulai siaran. Semua metrik dan peristiwa yang dihasilkan selama siaran dikaitkan dengan sumber daya saluran.	✓		
Tipe saluran	Menentukan resolusi dan laju bingkai yang diizinkan untuk saluran. Lihat Jenis Saluran di IVS Referensi Streaming API Latensi Rendah.	✓		
Pembuatan log obrolan	Sebuah opsi lanjutan yang dapat diaktifkan dengan mengaitkan konfigurasi pembuatan log dengan ruang obrolan.			✓
Ruang obrolan	IVSSumber daya yang menyimpan konfigurasi untuk sesi obrolan, termasuk fitur opsional seperti Message Review Handler dan Chat Logging. Lihat Langkah 2: Buat Ruang Obrolan di Memulai IVS Obrolan.			✓
Komposisi di sisi klien	Menggunakan perangkat host untuk mencampur aliran audio dan video dari peserta panggung dan kemudian mengirimkannya sebagai aliran komposit ke IVS saluran. Hal ini memungkinkan kontrol yang lebih leluasa atas tampilan komposisi dengan mengorbankan pemanfaatan sumber daya klien yang lebih tinggi dan risiko yang lebih tinggi atas masalah stage atau host yang berdampak pada pemirsa. Lihat juga komposisi di sisi server.	✓	✓	
CloudFront	CDNLayanan yang disediakan oleh Amazon.	✓		

Istilah	Deskripsi	LL	RT	Obrolan
CloudTrail	AWSLayanan untuk mengumpulkan, memantau, menganalisis, dan mempertahankan peristiwa dan aktivitas akun dari AWS dan sumber eksternal. Lihat Pencatatan IVS API Panggilan dengan AWS CloudTrail.	✓	✓	✓
CloudWatch	AWSLayanan untuk memantau aplikasi, menanggapi perubahan kinerja, mengoptimalkan penggunaan sumber daya, dan memberikan wawasan tentang kesehatan operasional. Anda dapat menggunakannya CloudWatch untuk memantau IVS metrik; lihat Memantau Streaming IVS Waktu Nyata dan Memantau Streaming IVS Latensi Rendah.	✓	✓	✓
Komposisi	Proses menggabungkan streaming audio dan video dari lebih dari satu sumber menjadi satu streaming.	✓	✓	
Pipeline komposisi	Suatu urutan langkah-langkah pemrosesan yang diperlukan untuk menggabungkan banyak streaming dan mengenkode streaming yang dihasilkan.	✓	✓	
Kompresi	Proses enkode informasi dengan menggunak an bit yang lebih kecil dari representasi asli. Kompresi tertentu dapat bersifat lossless atau lossy. Kompresi lossless mengurangi bit dengan mengidentifikasi dan menghilangkan redundans i statistik. Tidak ada informasi yang hilang dalam kompresi lossless. Kompresi lossy mengurangi bit dengan menghapus informasi yang tidak perlu atau tidak terlalu penting.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
Bidang kontrol	Menyimpan informasi tentang IVS sumber daya seperti saluran, tahapan, atau ruang obrolan dan menyediakan antarmuka untuk membuat dan mengelola sumber daya ini. Ini regional (berdasar kan AWS wilayah).	✓	✓	✓
CORS	Cross-Origin Resource Sharing, AWS fitur yang memungkinkan aplikasi web klien yang dimuat dalam satu domain untuk berintera ksi dengan sumber daya seperti bucket S3 di domain yang berbeda. Akses dapat dikonfigu rasi berdasarkan header, HTTP metode, dan domain asal. Lihat Menggunakan berbagi sumber daya lintas asal (CORS) - Layanan Penyimpan an Sederhana Amazon di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.	✓		
Sumber gambar kustom	Antarmuka yang disediakan oleh IVS Broadcast SDKyang memungkinkan aplikasi untuk memberika n input gambarnya sendiri alih-alih terbatas pada kamera preset.	✓	✓	
Bidang data	Infrastruktur yang membawa data dari ingest ke egress. Ini beroperasi berdasarkan konfigurasi yang dikelola di bidang kontrol dan tidak terbatas pada suatu AWS wilayah.	✓	✓	✓
Enkoder, enkode	Proses konversi konten video dan audio ke dalam format digital, cocok untuk streaming. Enkode dapat berbasis perangkat keras atau perangkat lunak.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
Peristiwa	Pemberitahuan otomatis yang diterbitkan oleh IVS ke layanan AmazonEventBridge pemantaua n. Suatu peristiwa merepresentasikan perubahan status atau kondisi dari sumber daya streaming seperti stage atau pipeline komposisi. Lihat Menggunakan Amazon EventBridge dengan Streaming IVS Latensi Rendah dan Menggunakan Amazon EventBridge dengan Streaming IVS Waktu Nyata.	✓	✓	✓
FFmpeg	Sebuah proyek perangkat lunak sumber terbuka dan gratis yang terdiri dari serangkaian pustaka serta program untuk menangani file dan streaming video serta audio. FFmpegmenyediakan solusi lintas platform untuk merekam, mengonversi, dan mengalirkan audio dan video.	✓		
Aliran terfragme ntasi	Dibuat ketika siaran terputus kemudian terhubung kembali dalam interval yang ditentukan dalam konfigurasi perekaman saluran. Banyaknya streaming yang dihasilkan dianggap sebagai satu siaran dan digabungkan menjadi satu streaming rekaman tunggal. Lihat Menggabungkan Streaming Terfragmentasi di Rekam Otomatis ke Amazon S3 (Streaming Latensi Rendah).	✓		
Laju bingkai	Suatu metrik streaming untuk jumlah bingkai video yang ditransmisikan atau diterima per detik.	✓	✓	
HLS	HTTPLive Streaming (HLS), protokol komunikas i streaming bitrate adaptif HTTP berbasis yang digunakan untuk mengirimkan IVS streaming ke pemirsa.	✓		

Istilah	Deskripsi	LL	RT	Obrolan
HLSdaftar putar	Daftar segmen media yang menyusun streaming . HLSDaftar putar standar terdiri dari file media 10 detik. HLSjuga mendukung daftar putar rentang byte yang lebih granular.	✓		
Host	Peserta acara waktu nyata yang mengirimkan video dan/atau audio ke stage.		✓	
IAM	Identity and Access Management, AWS layanan yang memungkinkan pengguna mengelola identitas dan akses ke AWS layanan dan sumber daya dengan aman, termasuk. IVS	✓	✓	✓
Ingest	IVSproses untuk menerima aliran video dari host atau penyiar untuk diproses atau dikirim ke pemirsa atau peserta lain.	✓	✓	
Ingest server	Menerima streaming video dan mengirimkannya ke sistem transcoding, di mana aliran ditransmu xed atau ditranskode ke dalam untuk pengiriman ke pemirsa. HLS Server Ingest adalah IVS komponen spesifik yang menerima aliran untuk saluran, bersama dengan protokol konsumsi (,). RTMP RTMPS Lihat informasi tentang membuat saluran di Memulai dengan Streaming IVS Latensi Rendah.		✓	
Video interlaced	Mentransmisikan dan menampilkan hanya baris ganjil atau genap dari bingkai berikutnya untuk menciptakan kesan penggandaan laju bingkai tanpa menghabiskan bandwidth ekstra. Kami tidak menyarankan menggunakan video interlaced terkait kualitas video.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
JSON	JavaScript Object Notation, format file standar terbuka yang menggunakan teks yang dapat dibaca manusia untuk mengirimkan objek data yang terdiri dari pasangan nilai atribut dan tipe data array atau nilai serialisasi lainnya.	✓	✓	✓
Keyframe, bingkai delta, interval keyframe	Keyframe (disebut juga dengan intra-coded atau i-frame) adalah bingkai penuh dari gambar dalam sebuah video. Bingkai berikutnya, bingkai delta (disebut juga dengan predicted atau p-frame), hanya berisi informasi yang telah berubah. Keyframe akan muncul berkali-kali dalam streaming, bergantung pada interval keyframe yang ditentukan dalam enkoder.	✓	✓	
Lambda	AWSLayanan untuk menjalankan kode (disebut sebagai fungsi Lambda) tanpa menyediakan infrastruktur server apa pun. Fungsi Lambda dapat berjalan sebagai respons terhadap peristiwa dan permintaan invokasi, atau berdasarkan jadwal. Misalnya, IVS Obrolan menggunakan fungsi Lambda untuk mengaktifkan peninjauan pesan untuk ruang obrolan.	✓	✓	✓
Latensi, glass-to-glass latensi	Keterlambatan dalam transfer data. IVSmendef inisikan rentang latensi sebagai: • Latensi rendah: di bawah 3 detik • Latensi waktu nyata: di bawah 300 ms lass-to-glassLatensi G mengacu pada penundaan dari saat kamera menangkap streaming langsung hingga saat streaming muncul di layar pemirsa.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
Enkode berlapis dengan simulcast	Memungkinkan enkode dan publikasi simultan untuk banyak streaming video dengan tingkat kualitas yang berbeda. Lihat Streaming Adaptif: Enkode Berlapis dengan Simulcast di Optimisasi Streaming Waktu Nyata.		✓	
Handler tinjauan pesan	Memungkinkan pelanggan IVS Chat untuk secara otomatis meninjau/memfilter pesan obrolan pengguna sebelum dikirim ke ruang obrolan. Handler ini diaktifkan dengan mengaitkan fungsi Lambda dengan ruang obrolan. Lihat Membuat Fungsi Lambda di Handler Tinjauan Pesan Obrolan.			✓
Mixer	Fitur Siaran IVS Seluler SDKsyang mengambil beberapa sumber audio dan video dan menghasil kan satu output. Fitur ini mendukung manajemen elemen video dan audio di layar yang merepresentasikan sumber-sumber seperti kamera, mikrofon, tangkapan layar, dan audio serta video yang dihasilkan oleh aplikasi. Output kemudian dapat dialirkan keIVS. Lihat Mengonfigurasi Sesi Siaran untuk Pencampuran dalam IVSSiaranSDK: Panduan Mixer (Streaming Latensi Rendah).	✓		
Streaming banyak host	Menggabungkan streaming dari banyak host ke dalam satu streaming. Hal ini dapat dicapai dengan menggunakan komposisi di sisi klien atau sisi server. Dengan streaming banyak host, banyak skenario yang dapat dilakukan, seperti mengundang pemirsa ke stage untuk Tanya Jawab, kompetisi antar-host, obrolan video, dan percakapan antara host di depan banyak pemirsa.		✓	

Istilah	Deskripsi	LL	RT	Obrolan
Daftar putar multivarian	Indeks dari semua streaming varian yang tersedia untuk siaran.	✓		
OAC	Origin Access Control, mekanisme untuk membatasi akses ke bucket S3, sehingga konten seperti aliran rekaman hanya dapat disajikan melalui. CloudFrontCDN	✓		
OBS	Open Broadcaster Software, perangkat lunak gratis dan sumber terbuka untuk perekaman video dan streaming langsung. OBSmenawarkan alternatif (untuk IVS siaran SDK) untuk penerbita n desktop. Streamer yang lebih canggih yang akrab dengannya OBS mungkin lebih menyukain ya karena fitur produksinya yang canggih, seperti transisi adegan, pencampuran audio, dan grafik overlay.	✓	✓	
Peserta	Pengguna waktu nyata yang terhubung ke stage sebagai host atau pemirsa.		✓	
Token peserta	Mengautentikasi peserta acara waktu nyata saat mereka bergabung dengan suatu stage. Token peserta juga mengontrol apakah peserta dapat mengirim video ke stage.		✓	

Istilah	Deskripsi	LL	RT	Obrolan
Token pemutaran , pasangan kunci pemutaran	Mekanisme otorisasi yang memungkinkan pelanggan membatasi pemutaran video di saluran privat. Token pemutaran dihasilkan dari pasangan kunci pemutaran. Pasangan kunci pemutaran adalah pasangan kunci publik-privat yang digunakan untuk menandata ngani dan memvalidasi token otorisasi pemirsa untuk pemutaran. Lihat Membuat atau Mengimpor Kunci IVS Pemutaran dalam Menyiapkan Saluran IVS Pribadi dan lihat titik akhir Pasangan Kunci Putar di IVSReferensi Latensi Rendah API.	✓		
Pemutaran URL	Mengidentifikasi alamat yang digunakan oleh pemirsa guna memulai pemutaran untuk saluran tertentu. Alamat ini dapat digunakan secara global. IVSsecara otomatis memilih lokasi terbaik di jaringan pengiriman konten IVS global untuk mengirimkan video ke setiap pemirsa. Lihat informasi tentang membuat saluran di Memulai dengan Streaming IVS Latensi Rendah.	✓		
Saluran privat	Memungkinkan pelanggan membatasi akses ke streaming mereka dengan menggunakan mekanisme otorisasi berdasarkan token pemutaran . Lihat Alur Kerja untuk Saluran IVS Pribadi di Menyiapkan Saluran IVS Pribadi.	✓		
Video progresif	Mentransmisikan dan menampilkan semua baris dari setiap bingkai secara berurutan. Kami menyarankan penggunaan video progresif selama semua stage siaran.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
Kuota	Jumlah maksimum sumber daya IVS layanan atau operasi untuk AWS akun Anda. Artinya, batas ini per AWS akun, kecuali disebutkan sebaliknya. Semua kuota diberlakukan per wilayah. Lihat titik akhir dan kuota Layanan Video Interaktif Amazon di Panduan Referensi AWS Umum.	✓	✓	✓
Wilayah	Menyediakan akses ke AWS layanan yang secara fisik berada di wilayah geografis tertentu. Wilayah memberikan toleransi kesalahan, stabilitas, serta ketahanan, dan juga dapat mengurangi latensi. Dengan Wilayah, Anda dapat membuat sumber daya redundan yang tetap tersedia dan tidak terpengaruh oleh pemadaman wilayah. Sebagian besar permintaan AWS layanan dikaitkan dengan wilayah geografis tertentu. Sumber daya yang Anda buat di satu wilayah tidak ada di wilayah lain kecuali Anda secara eksplisit menggunakan fitur replikasi yang ditawarkan oleh layanan. AWS Misalnya, Amazon S3 mendukung replikasi lintas wilayah. Beberapa layanan, seperti IAM, tidak memiliki sumber daya lintas daerah.	✓	✓	✓
Resolusi	Menjelaskan jumlah piksel dalam satu bingkai video, misalnya, Full HD atau 1080p mendefini sikan sebuah bingkai dengan 1920 x 1080 piksel.	✓	✓	
Pengguna root	Pemilik AWS akun. Pengguna root memiliki akses lengkap ke semua AWS layanan dan sumber daya di AWS akun.	✓	✓	✓

Istilah	Deskripsi	LL	RT	Obrolan
RTMP, RTMPS	Protokol Perpesanan Waktu Nyata, sebuah standar industri untuk mentransmisikan audio, video, dan data melalui jaringan. RTMPSadalah versi amanRTMP, berjalan melalui koneksi Transport Layer Security (TLS/SSL).	✓	✓	
Bucket S3	Kumpulan objek yang disimpan di Amazon S3. Banyak kebijakan, termasuk akses dan replikasi, ditentukan pada tingkat bucket dan berlaku untuk semua objek di bucket. Misalnya, IVS siaran disimpan sebagai beberapa objek dalam ember S3.	✓		
SDK	Software Development Kit, kumpulan perpustak aan untuk pengembang membangun aplikasi denganIVS.	✓	✓	✓
Segmentasi selfie	Memungkinkan penggantian latar belakang dalam streaming langsung dengan menggunakan solusi khusus klien yang menerima gambar kamera sebagai input dan mengembalikan mask yang memberikan skor kepercayaan untuk setiap piksel gambar, yang menunjukkan apakah gambar itu di latar depan atau latar belakang. Lihat Pengganti an Latar Belakang di IVSSiaranSDK: Filter Kamera Pihak Ketiga (Streaming Waktu Nyata).		✓	
Penentuan versi semantik	Sebuah format versi dalam bentuk Major.Min or.Patch. Perbaikan bug yang tidak memengaruhi API kenaikan versi tambalan, API penambahan/ perubahan yang kompatibel ke belakang meningkat kan versi minor, dan perubahan yang tidak kompatibel ke belakang meningkatkan versi utama. API	✓	✓	✓

Istilah	Deskripsi	LL	RT	Obrolan
Komposisi di sisi server	Menggunakan IVS server untuk mencampur audio dan video dari peserta panggung dan kemudian mengirimkan video campuran ini ke IVS saluran untuk menjangkau audiens yang lebih besar atau menyimpannya dalam ember S3. Komposisi di sisi server mengurangi beban klien sehingga meningkatkan ketahanan siaran dan memungkin kan penggunaan bandwidth yang lebih efisien. Lihat juga komposisi di sisi klien.		✓	
Kuota layanan	AWSLayanan yang membantu Anda mengelola kuota untuk banyak AWS layanan dari satu lokasi. Selain mencari nilai kuota, Anda juga dapat meminta peningkatan kuota dari konsol Service Quotas.	✓	✓	✓
Peran terkait layanan	Jenis IAMperan unik yang ditautkan langsung ke AWS layanan. Peran terkait layanan dibuat secara otomatis oleh IVS dan menyertakan semua izin yang diperlukan layanan untuk memanggil AWS layanan lain atas nama Anda, misalnya, untuk mengakses bucket S3. Lihat Menggunakan Peran Tertaut Layanan untuk Keamanan IVS. IVS	✓		
Stage	IVSSumber daya yang mewakili ruang virtual tempat peserta acara real-time dapat bertukar video secara real time. Lihat Membuat Panggung dalam Memulai Streaming IVS Waktu Nyata.		✓	
Sesi stage	Dimulai ketika peserta pertama bergabung dengan stage dan berakhir beberapa menit setelah peserta terakhir berhenti memublikasikan ke stage. Suatu stage berdurasi panjang mungkin memiliki banyak sesi selama masa tayangnya.		✓	

Istilah	Deskripsi	LL	RT	Obrolan
Streaming	Data yang merepresentasikan konten video atau audio yang dikirim secara terus-menerus dari sumber ke tujuan.	✓	✓	
Kunci streaming	Pengidentifikasi yang ditetapkan oleh IVS saat Anda membuat saluran; itu digunakan untuk mengotorisasi streaming ke saluran. Perlakukan kunci streaming seperti suatu rahasia, karena siapa pun yang memilikinya dapat melakukan streaming ke saluran. Lihat Memulai Streaming IVS Latensi Rendah.	✓		
Kekurangan streaming	Penundaan atau penghentian pengiriman aliran keIVS. Itu terjadi ketika IVS tidak menerima jumlah bit yang diharapkan bahwa perangkat pengkodean yang diiklankan akan dikirim selama jangka waktu tertentu. Terjadinya kekurangan streaming menyebabkan peristiwa kekurangan streaming. Dari sudut pandang pemirsa, kekurangan streaming dapat tampak seperti video terputus-putus, mengalami buffering, atau freezing. Kekurangan streaming dapat berlangsung singkat (kurang dari 5 detik) atau lama (beberapa menit), bergantung pada situasi spesifik yang mengakibatkan kekurangan streaming. Lihat Apa itu Stream Kelaparan dalam Pemecahan Masalah. FAQ	✓	✓	
Streamer	Seseorang atau perangkat yang mengirim streaming video atau audio keIVS.	✓	✓	
Pelanggan	Peserta acara waktu nyata yang menerima video dan/atau audio dari host. Lihat Apa itu Streaming IVS Waktu Nyata.		✓	

Istilah	Deskripsi	LL	RT	Obrolan
Tag	Label metadata yang Anda tetapkan ke sumber daya. AWS Tag dapat membantu Anda mengident ifikasi dan mengatur AWS sumber daya Anda. Pada halaman landing IVS dokumentasi, lihat “Penandaan” di IVS API dokumentasi mana pun (untuk streaming real-time, streaming latensi rendah, atau obrolan).	✓	✓	✓
Filter kamera pihak ketiga	Komponen perangkat lunak yang dapat diintegra sikan dengan IVS Broadcast SDKuntuk memungkin kan aplikasi memproses gambar sebelum memberikannya ke Broadcast SDK sebagai sumber gambar khusus. Sebuah filter kamera pihak ketiga dapat memproses gambar dari kamera, menerapkan efek filter, dll.	✓	✓	
Thumbnail	Sebuah gambar berukuran kecil yang diambil dari streaming. Secara default, thumbnail dihasilkan setiap 60 detik, tetapi interval yang lebih pendek juga dapat dikonfigurasi. Resolusi thumbnail bergantung pada tipe saluran. Lihat Merekam Konten di Rekam Otomatis ke Amazon S3 (Streaming Latensi Rendah).	✓		

Istilah	Deskripsi	LL	RT	Obrolan
Metadata berwaktu	Metadata yang terikat pada stempel waktu tertentu dalam suatu aliran. Hal ini dapat ditambahkan secara terprogram menggunakan IVS API dan menjadi terkait dengan frame tertentu. Hal ini memastikan bahwa semua pemirsa menerima metadata pada titik yang relatif sama terhadap streaming. Metadata berwaktu dapat digunakan untuk memicu tindakan pada klien seperti memperbarui statistik tim selama acara olahraga. Lihat Menyematkan Metadata dalam Streaming Video.	✓		
Transkode	Mengonversi video dan audio dari satu format ke format lainnya. Streaming masuk dapat ditransko de ke berbagai format pada berbagai laju bit dan resolusi, untuk mendukung berbagai perangkat pemutaran dan kondisi jaringan.	✓	✓	
Transmuxing	Pengemasan ulang sederhana dari aliran yang dicerna keIVS, tanpa pengkodean ulang aliran video. “Transmux” adalah kependekan dari transcode multiplexing, sebuah proses yang mengubah format file audio dan/atau video sambil menyimpan beberapa atau semua streaming asli. Transmuxing mengonversi ke berbagai format kontainer tanpa mengubah isi file. Berbeda dari transkode.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
Streaming varian	Suatu set enkode siaran yang sama dalam berbagai tingkat kualitas. Setiap aliran varian dikodekan sebagai daftar putar terpisahHLS. Indeks streaming varian yang tersedia disebut sebagai daftar putar multivarian. Setelah IVS pemain menerima daftar putar multivari an dariIVS, ia kemudian dapat memilih antara aliran varian selama pemutaran, berubah bolak-bal ik dengan mulus saat kondisi jaringan berubah.	✓		
VBR	Variable Bitrate, metode kontrol laju untuk enkoder yang menggunakan laju bit dinamis yang berubah sepanjang pemutaran, bergantung pada tingkat detail yang diperlukan. Kami sangat menyarankan untuk tidak menggunakan VBR karena masalah kualitas video; gunakan CBRsebagai gantinya.	✓	✓	
Tayang	Sesi penayangan unik yang secara aktif mengunduh atau memutar video. Tayangan adalah dasar untuk kuota tayangan bersamaan. Suatu tayangan dimulai ketika sesi penayangan memulai pemutaran video. Suatu tayangan diakhiri ketika sesi penayangan mengakhiri pemutaran video. Pemutaran adalah satu-satunya indikator pemirsa; heuristik interaksi seperti tingkat audio, fokus tab peramban, dan kualitas video bukan tidak dipertimbangkan. Saat menghitung penayangan, IVS tidak mempertimbangkan legitimasi pemirsa individu atau mencoba untuk menghapus duplikasi pemirsa lokal, seperti beberapa pemutar video pada satu mesin. Lihat Kuota Lainnya di Service Quotas (Streaming Latensi Rendah).	✓		
Pemirsa	Seseorang yang menerima aliran dariIVS.	✓		

Istilah	Deskripsi	LL	RT	Obrolan
Web RTC	Komunikasi Waktu Nyata Web, sebuah proyek sumber terbuka yang menyediakan peramban web dan aplikasi seluler dengan komunikasi waktu nyata. Ini memungkinkan komunikasi audio dan video bekerja di dalam halaman web dengan memungkinkan peer-to-peer komunikasi langsung, menghilangkan kebutuhan untuk menginstal plugin atau mengunduh aplikasi asli. Teknologi di balik Web RTC diimplementasikan sebagai standar web terbuka dan tersedia secara reguler JavaScript APIs di semua browser utama atau sebagai pustaka untuk klien asli, seperti Android dan iOS.	✓	✓	

Istilah	Deskripsi	LL	RT	Obrolan
WHIP	Web RTC - HTTP Ingestion Protocol, protokol HTTP berbasis yang memungkinkan konsumsi konten RTC berbasis Web ke dalam layanan streaming dan/atau. CDNs WHIPadalah IETF draf yang dikembangkan untuk membakukan konsumsi WebRTC. WHIPmemungkinkan kompatibilitas dengan perangkat lunak seperti OBS, menawarkan alternatif (untuk IVS siaran SDK) untuk penerbita n desktop. Streamer yang lebih canggih yang akrab dengannya OBS mungkin lebih menyukain ya karena fitur produksinya yang canggih, seperti transisi adegan, pencampuran audio, dan grafik overlay WHIPjuga bermanfaat dalam situasi di mana menggunakan IVS siaran SDK tidak layak atau disukai. Misalnya, dalam pengaturan yang melibatkan encoder perangkat keras, IVS siaran SDK mungkin bukan pilihan. Namun, jika encoder mendukungWHIP, Anda masih dapat mempublik asikan langsung dari encoder ke. IVS Lihat IVSWHIPSupport (Streaming Waktu Nyata).		✓	
WSS	WebSocket Aman, protokol untuk membangun WebSockets melalui koneksi terenkripsi. TLS Ini digunakan untuk menghubungkan ke titik akhir IVS Obrolan. Lihat Langkah 4: Kirim dan Terima Pesan Pertama Anda di Memulai IVS Obrolan.			✓

IVSRiwayat Dokumen Obrolan

Tabel berikut menjelaskan perubahan penting pada dokumentasi untuk IVS Obrolan Amazon. Kami sering memperbarui dokumentasi, untuk rilis baru dan untuk mengatasi umpan balik yang Anda kirimkan kepada kami.

Perubahan Panduan Pengguna Obrolan

Perubahan	Deskripsi	Tanggal
Pemecahan Panduan Pengguna Obrolan	Rilis ini disertai perubahan besar pada dokumenta si. Kami memindahkan informasi obrolan dari Panduan Pengguna Streaming IVS Latensi Rendah ke Panduan Pengguna IVS Obrolan baru, yang terletak di bagian IVS Obrolan yang ada di halaman arahan IVS dokumentasi. Untuk perubahan dokumenta si lainnya, lihat Riwayat Dokumen (Streaming Latensi Rendah).	28 Desember 2023
IVSGlosarium	Memperpanjang glosarium, mencakup istilah IVS real-time , latensi rendah, dan obrolan.	20 Desember 2023

IVSPerubahan API Referensi Obrolan

APIUbah	Deskripsi	Tanggal
Pemecahan Panduan Pengguna Obrolan	Sekarang setelah ada Panduan Pengguna IVS Obrolan (dibuat dalam rilis ini), entri Riwayat Dokumen untuk APIReferensi IVS Obrolan dan Referensi Pesan IVS API Obrolan yang ada akan ditemukan di sini, bergerak maju. Entri riwayat sebelumnya untuk API Referensi Obrolan tersebut ada di Riwayat Dokumen (Streaming Latensi Rendah).	28 Desember 2023

IVSCatatan Rilis Obrolan

Dokumen ini berisi semua catatan rilis IVS Obrolan Amazon, terbaru pertama, diatur berdasarkan tanggal rilis.

28 Desember 2023

Panduan Pengguna IVS Obrolan Amazon

Amazon Interactive Video Service (IVS) Chat adalah fitur obrolan langsung yang dikelola untuk mengikuti streaming video langsung. Dalam rilis ini, kami memindahkan informasi obrolan dari Panduan Pengguna Streaming IVS Latensi Rendah ke Panduan Pengguna IVS Obrolan baru. Dokumentasi dapat diakses dari halaman arahan IVS dokumentasi Amazon.

31 Januari 2023

Pesan Klien IVS Obrolan AmazonSDK: Android 1.1.0

Platform	Unduhan dan Perubahan
Pesan Klien Obrolan Android SDK 1.1.0	Dokumentasi referensi: https://aws.github.io/amazon-ivs-chat-messaging -sdk-android/1.1.0/ • Untuk mendukung Coroutine Kotlin, kami menambahkan Pesan IVS APIs Obrolan baru di paket com.amazonaws.ivs.chat.messaging.coroutines. Lihat juga tutorial Coroutine Kotlin yang baru; bagian 1 (dari 2) adalah Ruang Obrolan.

SDKUkuran Pesan Klien Obrolan: Android

Arsitektur	Ukuran Terkompresi	Ukuran Tidak Terkompresi
Semua arsitektur (bytecode)	89 KB	92 KB

9 November 2022

Pesan Klien IVS Obrolan AmazonSDK: JavaScript 1.0.2

Platform	Unduhan dan Perubahan
JavaScript Pesan Klien Obrolan SDK 1.0.2	Dokumentasi referensi: https://aws.github.io/amazon-ivs-chat-messaging -sdk-js/1.0.2/ • Memperbaiki masalah yang memengaruhi Firefox: klien secara keliru menerima kesalahan soket saat mereka terputus dari ruang obrolan menggunakan titik akhir. DisconnectUser

8 September 2022

Pesan Klien IVS Obrolan AmazonSDK: Android 1.0.0 dan iOS 1.0.0

Platform	Unduhan dan Perubahan
Pesan Klien Obrolan Android SDK 1.0.0	Dokumentasi referensi: https://aws.github.io/amazon-ivs-chat-messaging -sdk-android/1.0.0/
Pesan Klien Obrolan iOS SDK 1.0.0	Dokumentasi referensi: https://aws.github.io/amazon-ivs-chat-messaging -sdk-ios/1.0.0/

SDKUkuran Pesan Klien Obrolan: Android

Arsitektur	Ukuran Terkompresi	Ukuran Tidak Terkompresi
Semua arsitektur (bytecode)	53 KB	58 KB

SDKUkuran Pesan Klien Obrolan: iOS

Arsitektur	Ukuran Terkompresi	Ukuran Tidak Terkompresi
ios-arm64_x86_64-simulator (bitcode)	484 KB	2,4 MB
ios-arm64_x86_64-simulator	484 KB	2,4 MB
ios-arm64 (bitcode)	1,1 MB	3,1 MB
ios-arm64	233 KB	1,2 MB

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.