Chat

시나리오 솔루션

제품 문서

목록:

시나리오 솔루션
Live Room
라이브 룸 구축 가이드

최종 업데이트 날짜: : 2024-02-22 12:03:44

라이브 스트리밍이 보편화되면서 점점 더 많은 기업과 개발자들이 자체 라이브 스트리밍 플랫폼을 구축하고 있습니다. 이 과정에서 라이브 스트림 푸시 및 풀, 라이브 트랜스코딩, 라이브 화면 캡처, 라이브 스트림 믹싱, 라이브 채팅방, 라이브 룸 인터랙션(예: 좋아요, 선물 및 공동 앵커) 및 라이브 룸 상태 관리와 같은 복잡한 요구 사항을 처리해야 합니다. 이 문서에서는 Tencent Cloud 제품(Instant Messaging, Cloud Streaming Services)를 예로 들어 라이브 룸을 구현하는 방법과 가능한 문제 및 고려 사항을 설명하고, 라이브 스트리밍 비즈니스 및 요구 사항을 간략히 안내합니다.

1. 준비 작업

애플리케이션 생성

Tencent Cloud에서 라이브 룸을 설정하려면 아래와 같이 콘솔에서 IM 애플리케이션을 생성해야 합니다.

라이브 스트림 푸시 및 재생 도메인 추가

라이브 룸을 구축하려면 라이브 기능이 필수적이며 CSS를 통해 라이브 기능을 실현할 수 있습니다. 아래 그림과 같이 스트림 푸시 및 재생 도메인을 추가해야 합니다.

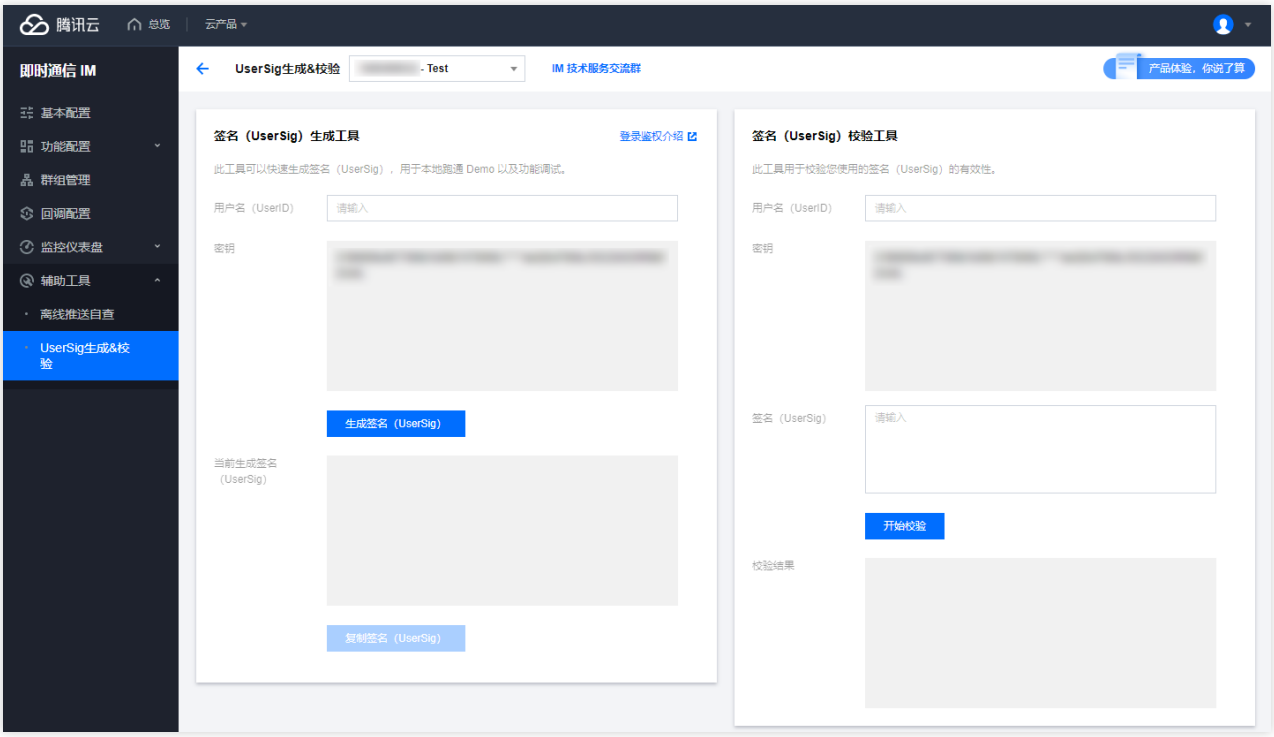
자세한 작업 방법은 외부 도메인 추가를 참고하십시오.

기본 구성 완료

준비 작업에서 생성된 애플리케이션은 개발에만 적용되는 무료 버전입니다. 프로덕션 환경에서는 필요에 따라 프로 또는 플래그십 에디션을 활성화해야 합니다. 버전별 차이점에 대한 자세한 내용은 요금 안내를 참고하십시오.
라이브 스트리밍 시나리오에서는 애플리케이션을 생성한 후 몇 가지 추가 구성이 필요합니다.

키로 UserSig 계산하기

IM 계정 시스템에서 사용자 로그인에 필요한 암호는 IM에서 제공한 키를 사용하여 서버에서 계산합니다. 자세한 내용은 Generating UserSig를 참고하십시오. 개발 단계에서 클라이언트의 개발 지연을 방지하기 위해 아래와 같이 콘솔에서 UserSig 계산할 수도 있습니다.

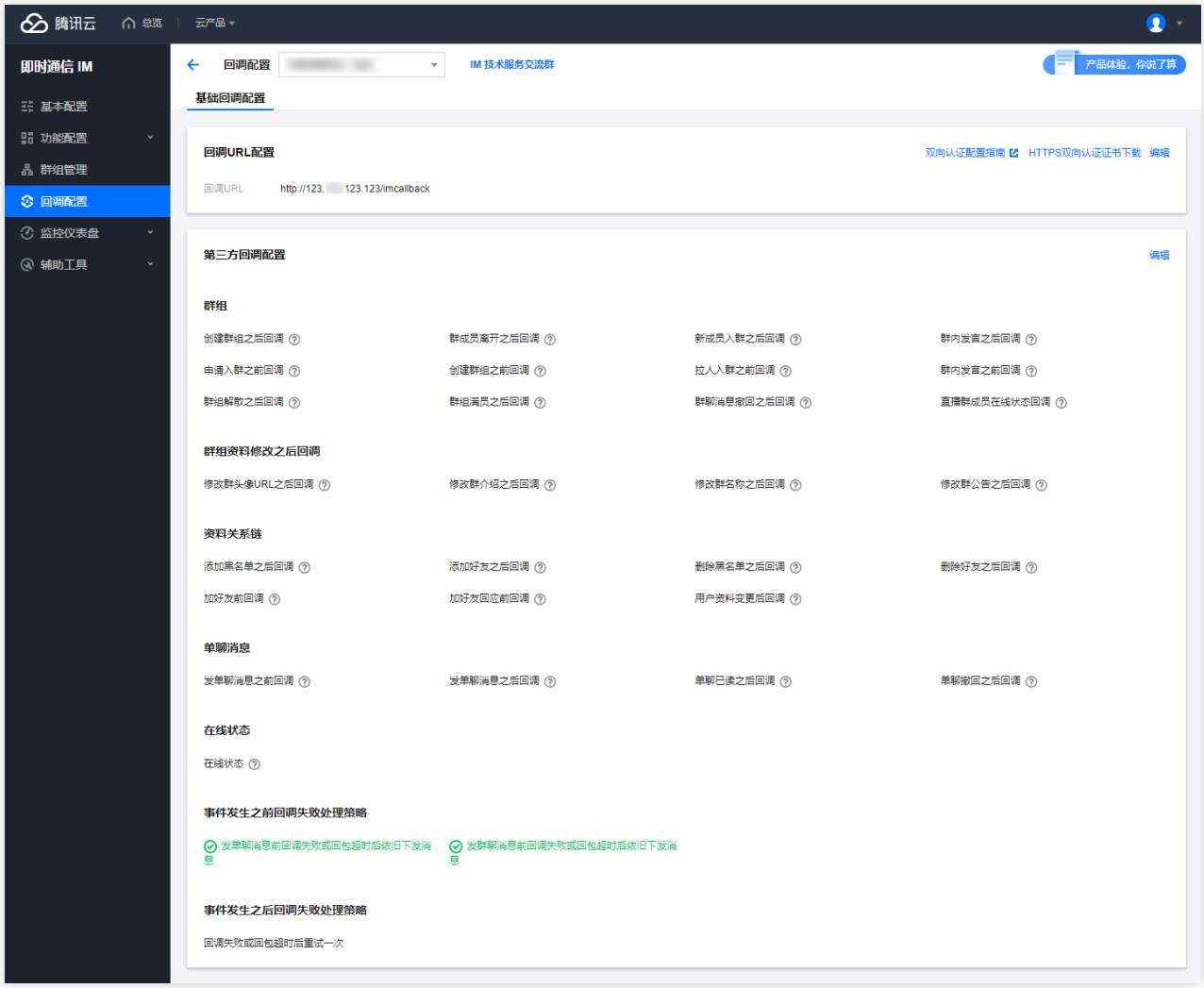


관리자 계정 구성

라이브 스트리밍 중에 관리자는 라이브 룸에 메시지를 보내거나 정책을 준수하지 않는 사용자를 음소거(강제 퇴장)해야 할 수 있습니다. 이 작업은 RESTful APIs를 통해 수행할 수 있습니다. 이러한 API를 호출하려면 IM 관리자 계정 생성을 해야 합니다. 기본적으로 IM은 UserID가 administrator인 계정을 제공합니다. 필요에 따라 여러 관리자 계정을 만들 수도 있으며, 최대 5개의 관리자 계정을 만들 수 있습니다.

콜백 주소 구성 및 콜백 활성화

라이브 룸에서 화면 댓글을 기반으로한 경품 추첨, 메시지 통계 수집, 민감한 콘텐츠 감지 등 요구 사항을 구현하려면 IM 백엔드가 특정 시나리오에서 비즈니스 백엔드를 다시 호출하는 IM 콜백 모듈을 사용해야 합니다. HTTP API를 제공하고 아래와 같이 콘솔 > 콜백 구성 모듈에서 구성하기만 하면 됩니다.



클라이언트 SDK 통합

준비가 끝나면 IM 및CSS 클라이언트 SDK를 프로젝트에 통합해야 합니다. 필요에 따라 다른 통합 옵션을 선택할 수 있습니다.

IM은 Instant Messaging 시작하기를 참고하십시오.

CSS는 MLVB SDK 시작하기를 참고하십시오.

다음은 라이브 룸의 일반적인 기능을 설명하고 구현 코드와 함께 모범 사례를 제공합니다.

2. 라이브 룸 기능 개발 가이드

라이브 룸 상태

라이브 룸의 상태는 다음과 같습니다.

No.	라이브 룸 상태
1	시작 예정

2	라이브 중
3	일시 중지
4	종료
5	재생됨

라이브 룸 상태에는 다음과 같은 특징이 있습니다.

No.	특징
1	라이브 룸에 있는 사용자에게 라이브 룸 상태 변경을 실시간으로 알려야 합니다.
2	라이브 룸을 처음 사용하는 사용자는 라이브 룸의 현재 상태를 확인해야 합니다.

두 가지 구현 스키마가 제공되며 각각의 장단점을 분석하면 다음과 같습니다.

구현 스키마	장점	단점
비즈니스 백엔드는 라이브 룸 상태를 유지하고 IM 서버 API를 사용하여 사용자에게 상태를 알리기 위해 Sending Ordinary Messages in a Group 합니다.	라이브 룸 상태를 자주 가져와야 하는 경우 비즈니스 백엔드에 상태를 저장하면 IM 그룹 프로필에 상태를 저장하는 것보다 IM SDK를 호출하는 빈도가 줄어듭니다. 이 구현 스키마를 사용하면 IM SDK가 통합되지 않은 경우 라이브 룸 상태를 얻을 수 있습니다.	비즈니스 백엔드는 라이브 룸 상태를 읽고 쓰기 위한 추가 모듈을 제공해야 합니다. 데이터는 비즈니스 백엔드와 IM 그룹 프로필 모두에서 가져오기 때문에 라이브 룸 데이터를 가져올 때 예외가 발생할 가능성이 더 큽니다. 사용자 정의 메시지를 보낼 때 손실될 수 있습니다. 많은 수의 메시지가 있는 경우 우선순위가 낮은 메시지가 먼저 삭제되어 라이브 룸 상태 표시에 영향을 줍니다. 따라서 우선순위가 높은 사용자 정의 메시지를 사용하는 것이 좋습니다.
사용자 정의 그룹 필드 또는 그룹 속성을 통해 라이브 룸 상태를 저장하고 클라이언트 SDK의 그룹 속성 변경 콜백을 통해 그룹 사용자에게 알릴 수 있습니다.	라이브 룸 상태를 읽고 쓰기 위한 추가 모듈을 제공할 필요가 없습니다. 이론적으로 그룹 속성 변경에 대한 콜백은 손실되지 않습니다. 라이브 룸 데이터는 예외를 줄이는 통합 데이터 소스 역할을 하는 그룹 프로필에서 가져옵니다.	고노출 모듈에서 그룹 프로필을 자주 가져와야 하므로 IM에 대한 부담이 커집니다. IM SDK 모듈이 통합되지 않은 경우 비즈니스 백엔드에서 IM 서버 SDK를 호출하여 그룹 프로필을 가져와야 하며 호출 수가 제한됩니다.

상기 분석을 바탕으로 라이브 룸 상태를 유지하기 위해 다음 두 가지 스키마를 결합할 것을 권장합니다.
1. 라이브 룸에서 스키마1을 사용하여 라이브 룸 상태를 가져옵니다.
2. 라이브 룸 외부에서 스키마2를 사용하여 라이브 룸 상태를 가져옵니다.
3. 비즈니스 백엔드에서 라이브 룸 상태가 변경된 후 RESTful APIs를 사용하여 데이터를 IM 그룹 프로필(그룹 속성 및 사용자 정의 그룹 필드)에 즉시 동기화합니다.

그룹 유형 선택

라이브 스트리밍 시나리오의 사용자 채팅 섹션에는 다음과 같은 특징이 있습니다.
1. 사용자가 자주 그룹에 가입하고 탈퇴하며 그룹 대화 정보(읽지 않은 횟수 및 lastMessage)를 관리할 필요가 없습니다.
2. 사용자는 승인 없이 그룹에 가입할 수 있습니다.
3. 사용자는 채팅 기록에 신경 쓰지 않고 메시지를 보냅니다.
4. 일반적으로 많은 수의 그룹 구성원이 있습니다.
5. 그룹 구성원 정보를 저장할 필요가 없습니다.
따라서 IM의 그룹 특성에 따라 라이브 룸의 그룹 유형으로 오디오/비디오 그룹(AVChatRoom)을 선택할 수 있습니다.
IM 오디오 비디오 그룹(AVChatRoom)은 다음과 같은 특징이 있습니다.
인원 제한이 없으며, 천만 규모의 인터랙티브 라이브 방송 시나리오를 구현할 수 있습니다.
모든 온라인 사용자 대상 푸시 메시지(그룹 시스템 알림)를 지원합니다.
그룹 참여 신청 후 관리자의 승인 없이 바로 참여할 수 있습니다.
설명：
Web용 IM SDK를 사용하면 사용자가 한 번에 하나의 오디오/비디오 그룹(AVChatRoom)에만 참여할 수 있습니다. 사용자가 클라이언트에 로그인하여 라이브 룸 A에 입장하면 콘솔에서 멀티 클라이언트 로그인이 활성화되고 사용자가 다른 클라이언트에 로그인하여 라이브 룸 B에 입장하면 해당 사용자는 라이브 룸 A에서 제거됩니다.

라이브 룸 공지

라이브 룸 공지(토픽)는 각 라이브 룸마다 필요하며, 방에 입장한 후 사용자가 볼 수 있습니다. 또한, 오디오/비디오 그룹 구성원에게 실시간으로 공지 변경 사항을 알려야 합니다. 라이브 룸 상태와 마찬가지로 비즈니스 백엔드 또는 IM 그룹 프로필에 라이브 룸 알림을 저장할 수 있습니다. 그러나 주의가 필요한 몇 가지 추가 사항이 있습니다.
1. 그룹 프로필의 그룹 notification 및 그룹 introduction 필드는 각각 최대 **240**바이트와 **300**바이트를 포함할 수 있습니다.
2. 그룹 프로필의 그룹 notification 및 그룹 introduction 필드는 string 유형만 지원합니다. 이미지와 텍스트로 알림을 생성하려면 json string 유형을 사용할 수 있으며 이미지는 온라인 url에 액세스할 수 있어야 합니다.
3. Web에서 editor를 통해 알림을 설정하는 경우, 보다 구체적으로 html 태그를 포함하는 경우 native에서 구문 분석되지 않을 수 있습니다.
라이브룸 알림은 RESTful APIs 또는 클라이언트 SDK의 그룹 속성을 통해 설정할 수 있습니다. 클라이언트는 그룹 속성을 통해 현재 그룹 정보를 얻고 GroupListener에서 OnGroupInfoChange를 수신하여 변경된 그룹 속성을 가져옵니다.

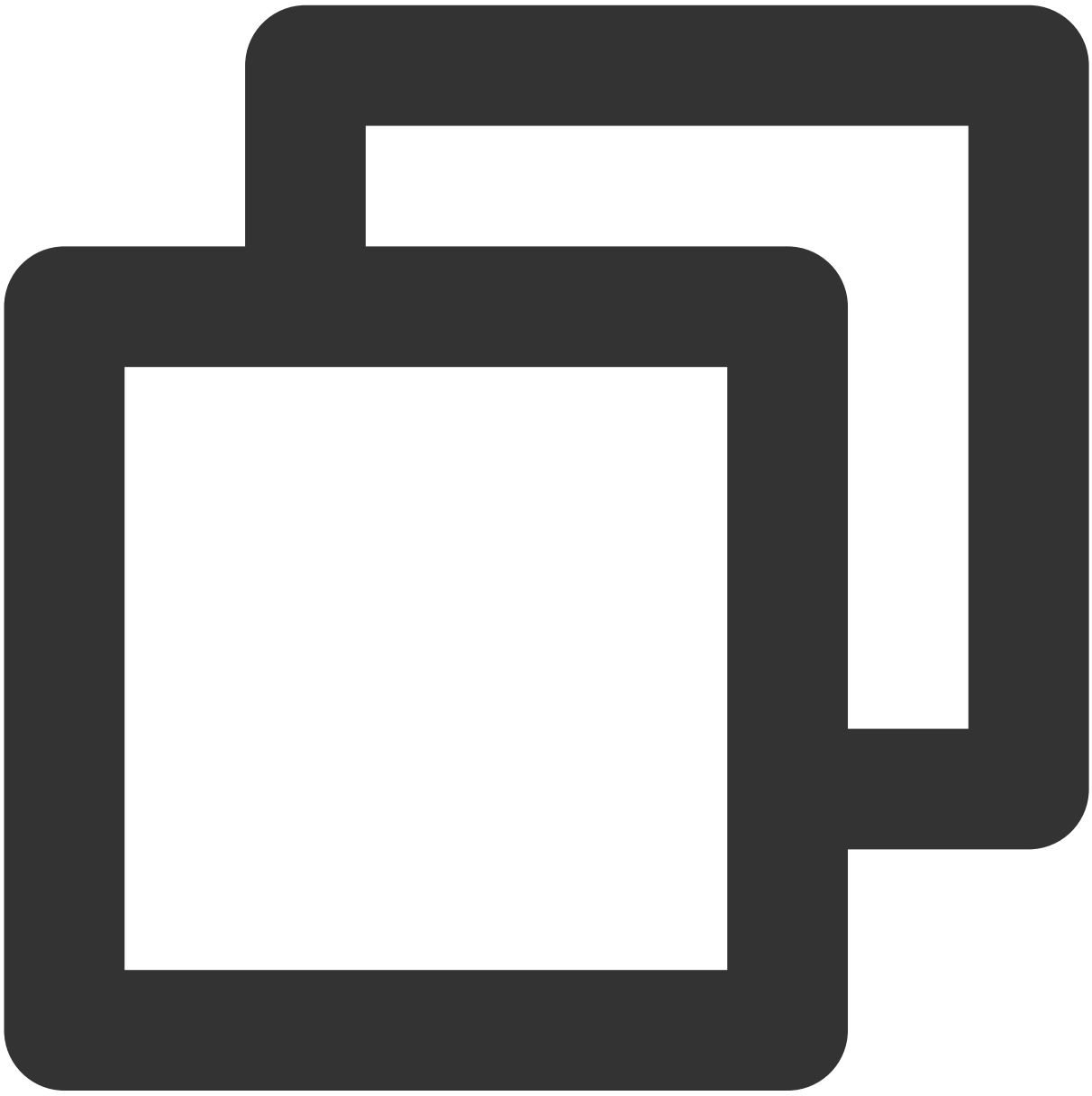
예시 코드:

Android

iOS & Mac

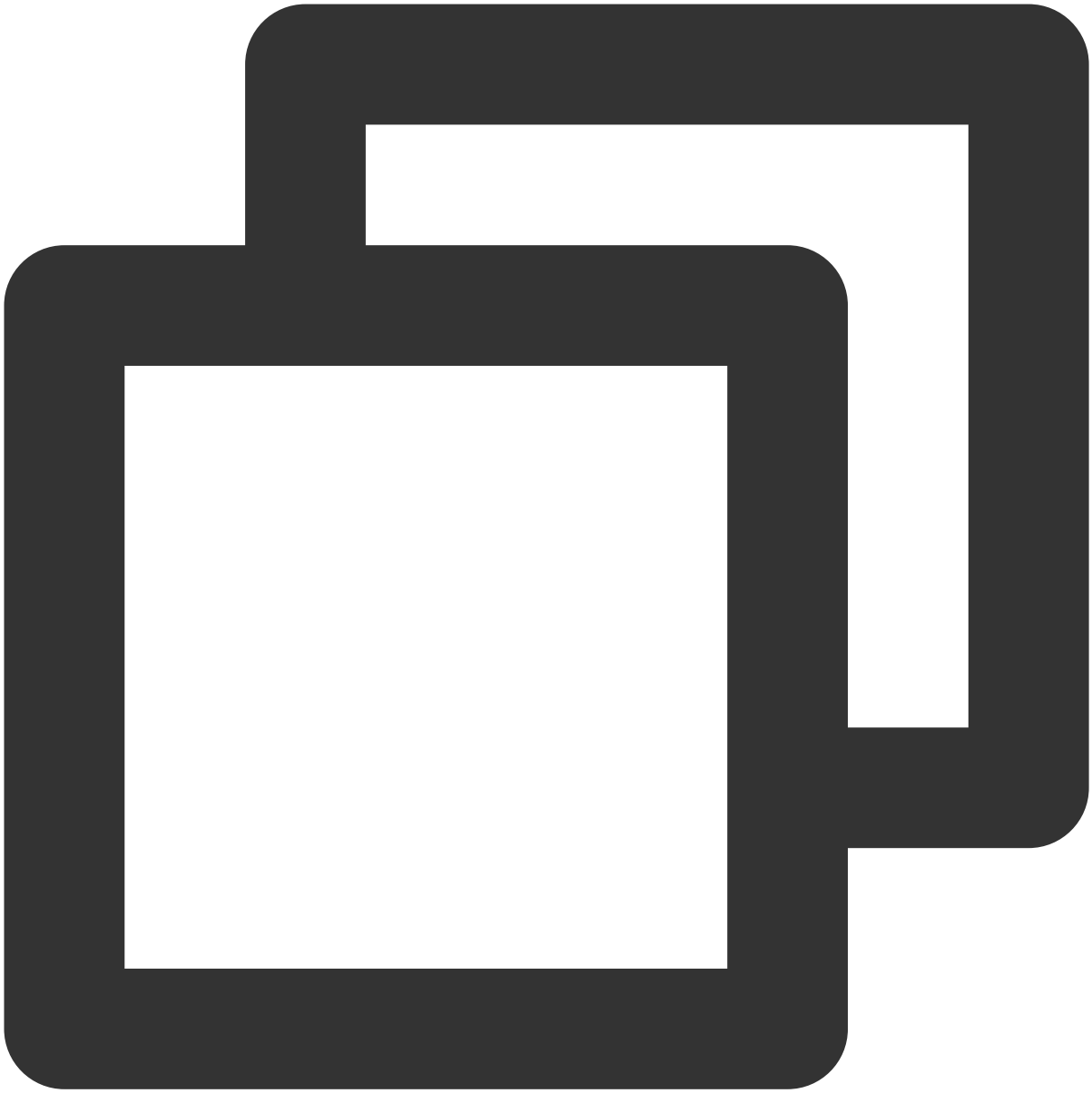
Flutter

Web



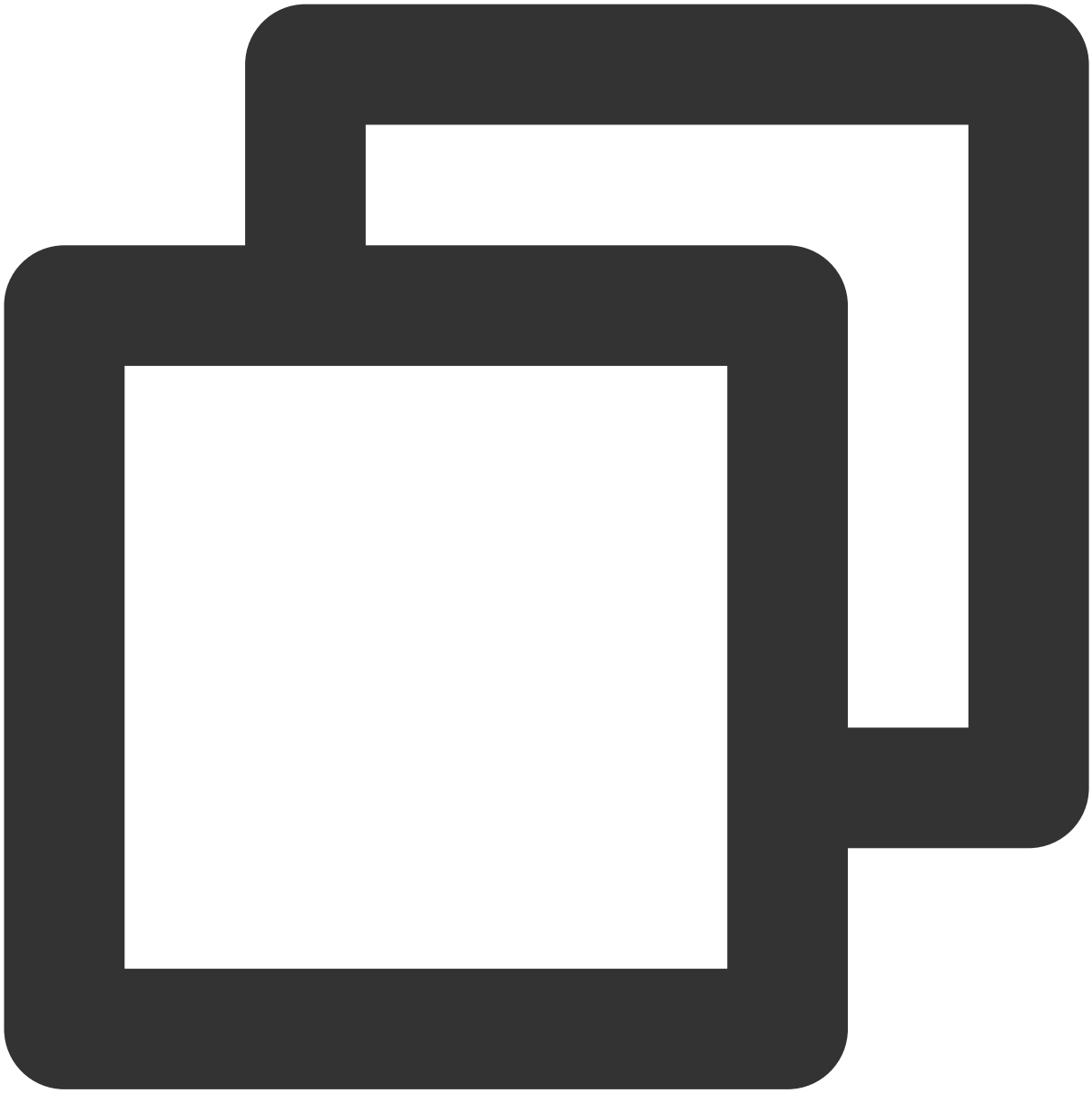
```
// 클라이언트에서 그룹 프로필 가져오기
V2TIMManager.getGroupManager().getGroupsInfo(groupIDList, new V2TIMValueCallback<Li
  @Override
  public void onSuccess(List<V2TIMGroupInfoResult> v2TIMGroupInfoResults) {
```

```
        // 그룹 프로필 가져오기 완료
    }

    @Override
    public void onError(int code, String desc) {
        // 그룹 프로필 가져오기 실패
    }
});
// 클라이언트 그룹 프로필 수정
V2TIMGroupInfo v2TIMGroupInfo = new V2TIMGroupInfo();
v2TIMGroupInfo.setGroupID("수정할 그룹 ID");
v2TIMGroupInfo.setFaceUrl("http://xxxx");
V2TIMManager.getGroupManager().setGroupInfo(v2TIMGroupInfo, new V2TIMCallback() {
    @Override
    public void onSuccess() {
        // 그룹 프로필 수정 완료
    }

    @Override
    public void onError(int code, String desc) {
        // 그룹 프로필 수정 실패
    }
});
```



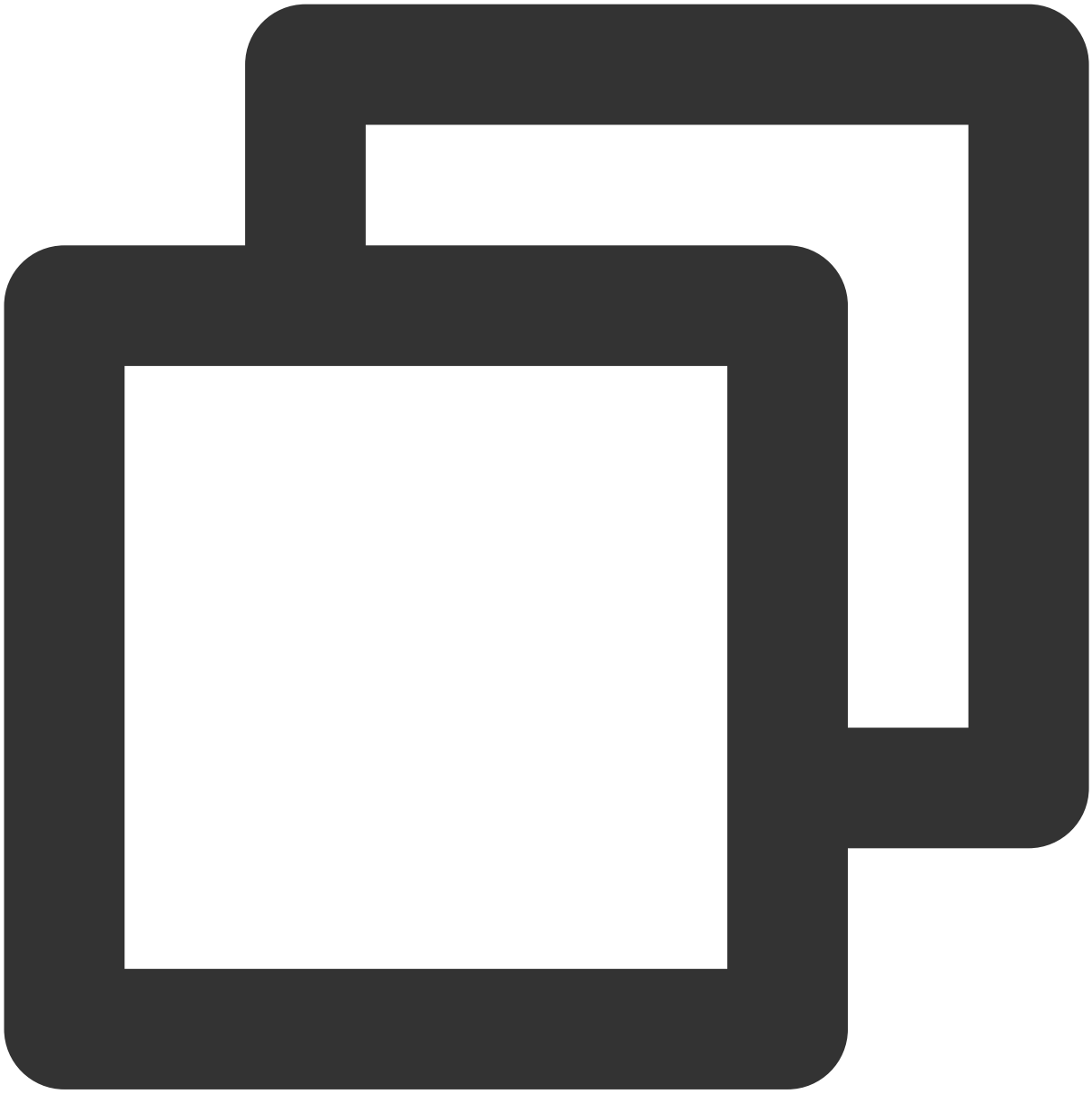
```
[[V2TIMManager sharedInstance] getGroupsInfo:@[@"groupA"] succ:^(NSArray<V2TIMGroup
    // 그룹 프로필 가져오기 완료
} fail:^(int code, NSString *desc) {
    // 그룹 프로필 가져오기 실패
}];
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"수정할 그룹 ID";
info.faceURL = @"http://xxxx";
[[V2TIMManager sharedInstance] setGroupInfo:info succ:^{
    // 그룹 프로필 수정 완료
} fail:^(int code, NSString *desc) {
```

```
    // 그룹 프로필 수정 실패
}];
```



```
// 그룹 프로필 가져오기
V2TimValueCallback<List<V2TimGroupInfoResult>> groupinfos = await groupManager.getG
// 그룹 프로필 수정
groupManager.setGroupInfo(info: V2TimGroupInfo.fromJson({
    "groupAddOpt":GroupAddOptTypeEnum.V2TIM_GROUP_ADD_AUTH
    // ... 기타 프로필
  }));
// 콜백
```

```
TencentImSDKPlugin.v2TIMManager.addGroupListener(listener: V2TimGroupListener(onGro
    // 그룹 정보 변경 콜백
  }));
```



```
// 그룹 프로필 가져오기
let promise = tim.getGroupProfile({ groupID: 'group1', groupCustomFieldFilter: ['ke
promise.then(function(imResponse) {
  console.log(imResponse.data.group);
}).catch(function(imError) {
  console.warn('getGroupProfile error:', imError); // 세부 그룹 프로필 가져오기 실패 정보
});
```

```
// 그룹 프로필 수정
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  name: 'new name', // 그룹 이름 수정
  introduction: 'this is introduction.', // 그룹 소개 수정
  // v2.6.0부터 그룹 구성원은 그룹 사용자 정의 필드 변경에 대한 그룹 팁 수신과 콘텐츠 획득, 자
  groupCustomField: [{ key: 'group_level', value: 'high'}] // 그룹 사용자 정의 필드 수
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 수정 완료 후 세부 그룹 프로필
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // 그룹 프로필 수정 실패 정보
});
// v2.6.2부터 SDK는 모두 음소거 및 음소거 해제를 지원합니다. 현재 그룹 내 모든 구성원이 음소거
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  muteAllMembers: true, // true: 전원 음소거, false: 전원 음소거 해제
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 수정 완료 후 세부 그룹 프로필
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // 그룹 프로필 수정 실패 정보
});
```

다른 SDK의 그룹 프로필 처리 모듈 예시 코드

메시지 우선순위

라이브 룸에는 많은 수의 사용자 메시지가 있습니다. 각 사용자는 자주 메시지를 보낼 수 있습니다. 전송된 메시지 수가 IM 빈도 제어 임계값에 도달하면 IM 백엔드는 시스템 실행 안정성을 보장하기 위해 일부 메시지를 삭제합니다. 클라이언트의 메시지 수신 빈도가 너무 높으면 메시지의 가독성이 떨어질 수 있으므로 이러한 임계값을 적용해야 합니다.

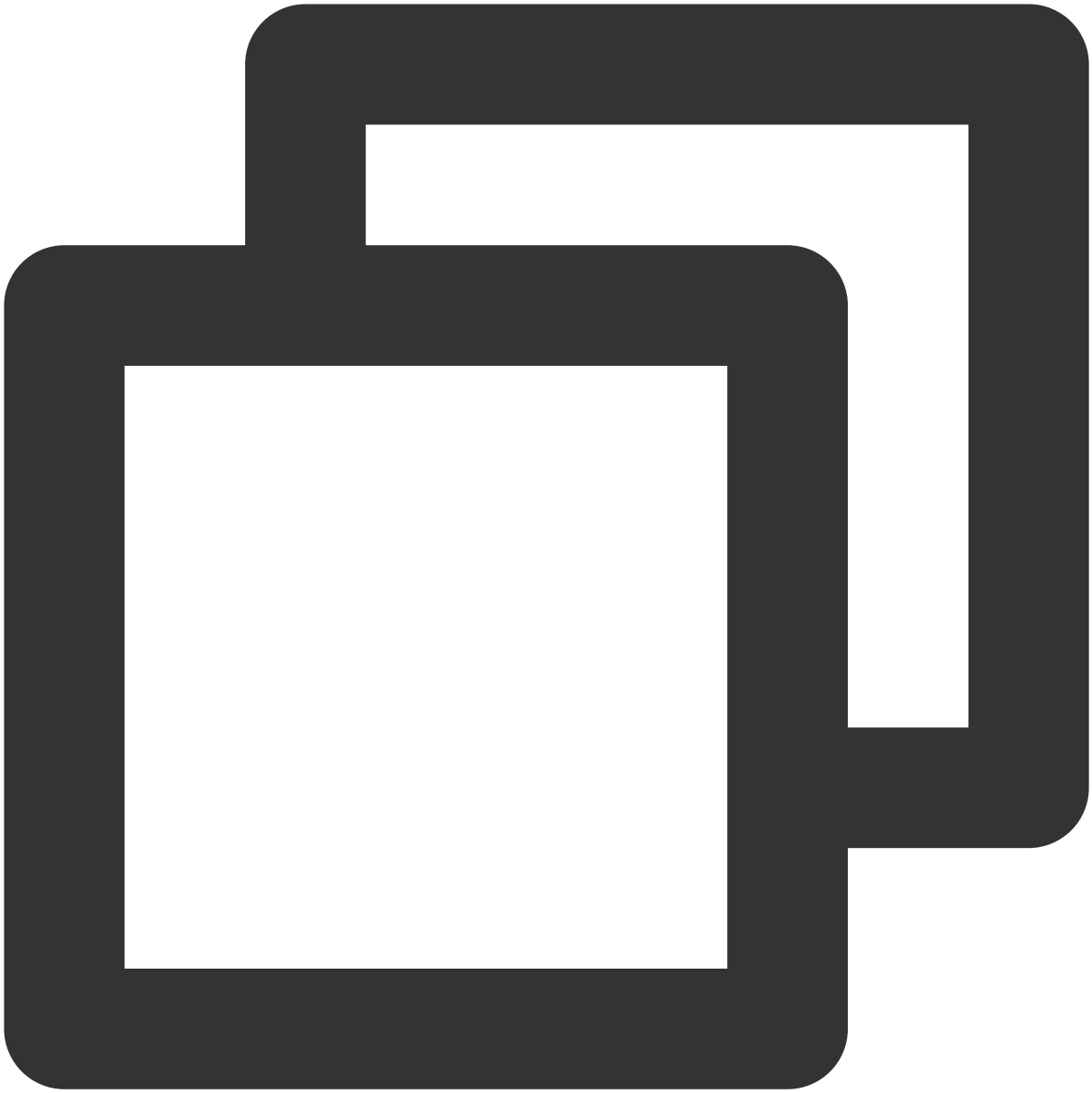
그룹 메시지에는 세 가지 우선순위가 있으며 백엔드는 우선순위가 높은 메시지를 먼저 전달합니다. 따라서 중요도에 따라 메시지에 대해 다른 우선순위를 설정하십시오.

다음은 3가지 우선순위를 높음에서 낮음 순으로 나열한 것입니다.

우선순위	설명
High	높은 우선순위
Normal	중간 우선순위
Low	낮은 우선순위

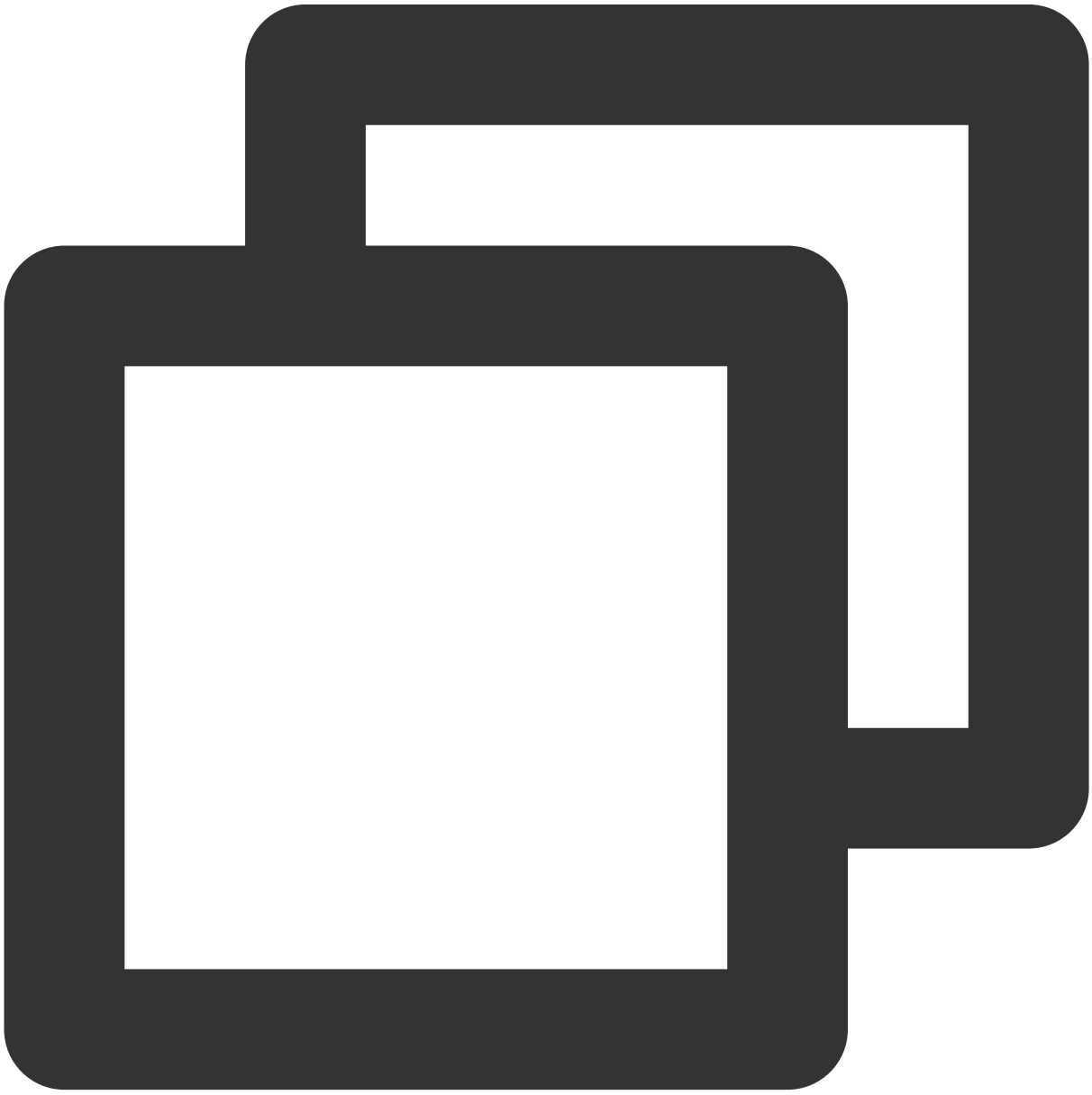
메시지는 다음 정책에 따라 삭제됩니다.

총 메시지 수에 대한 빈도 제어는 그룹에서 초당 보낼 수 있는 메시지 수를 제한합니다. 기본적으로 값은 40건/초입니다. 값을 초과하면 백엔드에서 우선순위가 높은 메시지를 먼저 전달하고 우선순위가 같은 메시지를 랜덤으로 전달합니다.
빈도 제어에 의해 제한된 메시지는 메시지 기록에 저장되지 않지만, 발신자에게 발송 성공이 반환될 수 있습니다. 이때, Callback Before Sending a Group Message가 트리거되지만 Callback After Sending a Group Message는 트리거되지 않습니다.
따라서 라이브 룸에 대한 메시지 우선순위를 설정해야 합니다.
라이브 룸 메시지는 다음과 같이 중요도에 따라 우선순위가 지정됩니다.
1. 모든 사용자에게 표시되어야 하는 보상 및 선물 메시지
2. 텍스트, 오디오 및 이미지 메시지
3. 좋아요 메시지
설명：
일반적으로 C 사용자가 보낸 메시지는 우선순위에 관계없이 화면에 표시됩니다. 많은 수의 메시지가 있는 경우 사용자가 다른 사용자의 메시지 일부가 손실되는 것이 허용됩니다.
메시지 우선순위 설정 예시 코드:
Android
iOS&Mac
Flutter
Web

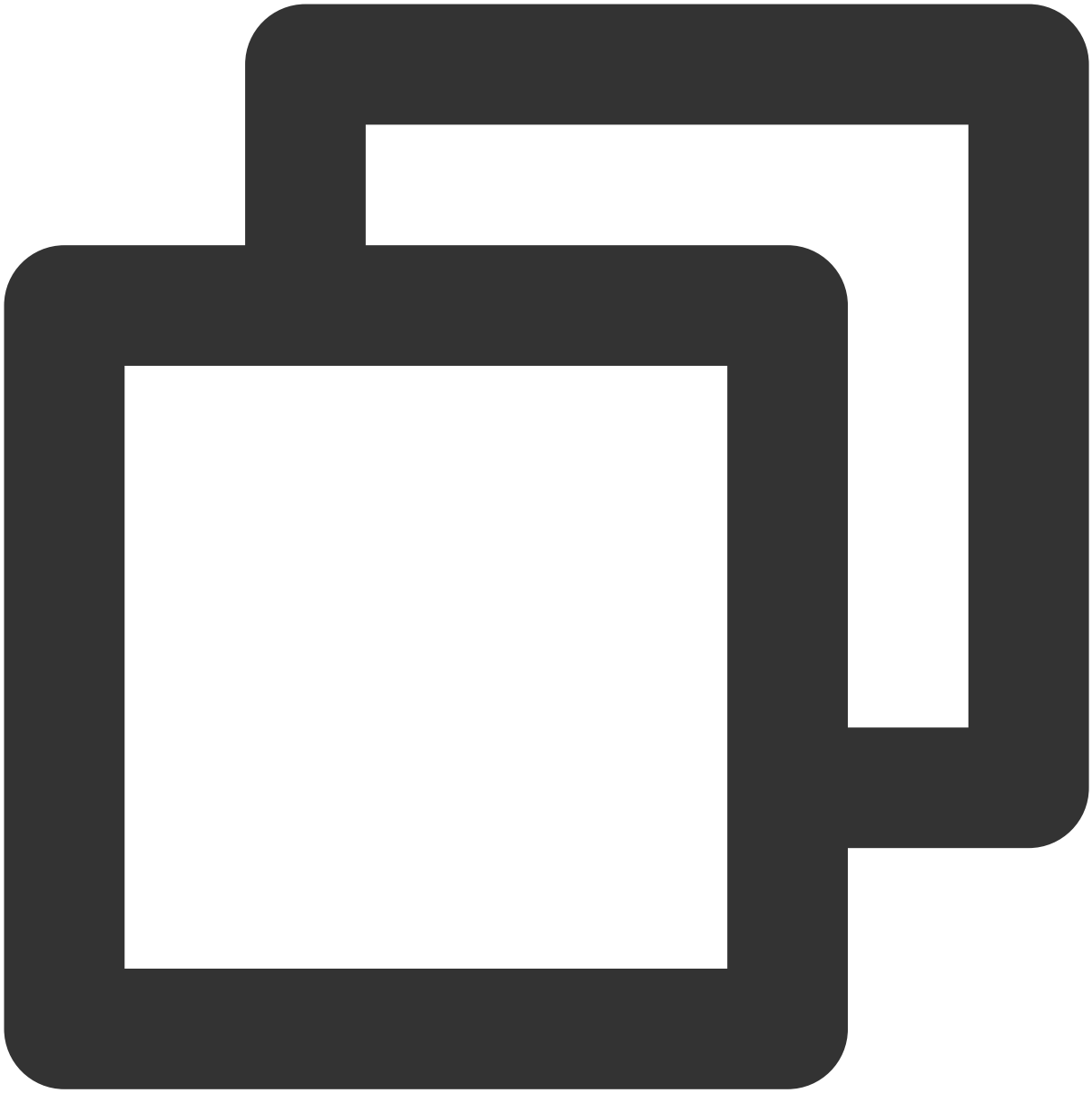


```
// 사용자 정의 메시지 생성
V2TIMMessage v2TIMMessage = V2TIMManager.getMessageManager().createCustomMessage("사
// 메시지 우선순위를 높은 우선순위 메시지로 설정
v2TIMMessage.setPriority(V2TIMMessage.V2TIM_PRIORITY_HIGH)
// 메시지 발송
V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, "receiver_userID", null,
    @Override
    public void onProgress(int progress) {
        // 사용자 정의 메시지에 대한 진행률은 다시 호출되지 않음
    }
```

```
    @Override
    public void onSuccess(V2TIMMessage message) {
        // 사용자 정의 그룹 메시지 전송 완료
    }

    @Override
    public void onError(int code, String desc) {
        // 사용자 정의 그룹 메시지 전송 실패
    }
});
```

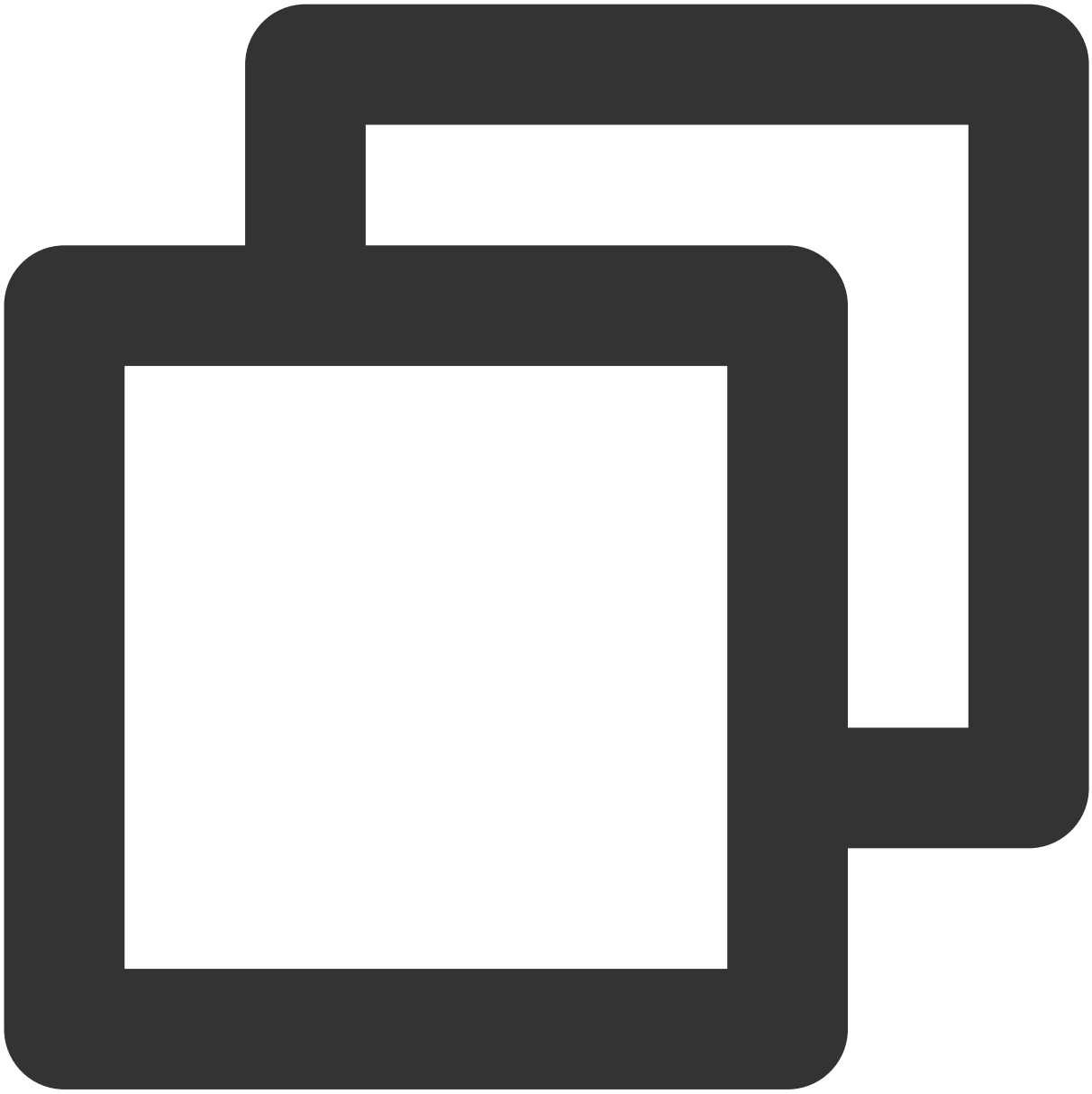


```
/ 텍스트 메시지 생성
V2TIMMessage *message = [[V2TIMManager sharedInstance] createTextMessage:@"content"
// 메시지 발송
[V2TIMManager.sharedInstance sendMessage:message
                                receiver:@"userID"
                                 groupID:nil
                                priority:V2TIM_PRIORITY_NORMAL
                          onlineUserOnly:NO
                         offlinePushInfo:nil
                                progress:nil
                                succ:^{
    // 텍스트 메시지 전송 완료
}
                                fail:^(int code, NSString *desc) {
    // 텍스트 메시지 전송 실패
}];
```



```
/ 텍스트 메시지 생성
V2TimValueCallback<V2TimMsgCreateInfoResult> createTextAtMessageRes = await Tencent
    text: "test",
    atUserList: [],
  );
 if(createTextAtMessageRes.code == 0){
    String id =  createTextAtMessageRes.data.id;

       // 텍스트 메시지 발송
    V2TimValueCallback<V2TimMessage> sendMessageRes = await TencentImSDKPlugin.v2TI
    if(sendMessageRes.code == 0){
```

```
      // 전송 완료
    }
  }
```



```
// 텍스트 메시지 발송
// 1. 메시지 인스턴스 생성, 반환된 인스턴스 화면에 표시 가능
let message = tim.createTextMessage({
  to: 'user1',
  conversationType: TIM.TYPES.CONV_C2C,
  // 그룹 채팅에 적용되는 메시지 우선순위(v2.4.2 이상에서 지원). 그룹의 메시지가 빈도 제한을 초
  // 지원되는 열거 값: TIM.TYPES.MSG_PRIORITY_HIGH, TIM.TYPES.MSG_PRIORITY_NORMAL(기본
```

```
    // priority: TIM.TYPES.MSG_PRIORITY_NORMAL,
    payload: {
      text: 'Hello world!'
    },
    // v2.20.0 부터 일대일 메시지에 대해 수신 확인 기능이 지원됩니다. 사용하려면 플래그십 에디션
    needReadReceipt: true
    // 클라우드에 저장된 사용자 정의 메시지 데이터 (수신자에게 전송되며 애플리케이션을 제거하고 다
    // cloudCustomData: 'your cloud custom data'
});
// 2. 메시지 전송
let promise = tim.sendMessage(message);
promise.then(function(imResponse) {
    // 전송 완료
    console.log(imResponse);
}).catch(function(imError) {
    // 전송 실패
    console.warn('sendMessage error:', imError);
});
```

다른 SDK의 메시지 우선순위 설정 예시 코드

선물 및 좋아요 메시지 모범 사례

선물 메시지

1. 클라이언트의 비영구적 연결 요청은 과금 로직과 관련된 비즈니스 서버로 전송됩니다.
2. 요금이 발생한 후 발신자는 XXX가 XXX 선물을 보낸 것을 볼 수 있습니다. (보내는 사람이 자신이 보낸 선물을 볼 수 있도록, 메시지가 많은 경우 메시지 폐기 정책이 트리거될 수 있음)

3. 요금 정산 후 서버 API를 호출하여 사용자 정의 메시지(선물)를 보낼 수 있습니다.
4. 여러 개의 선물을 연속으로 보낼 경우 메시지를 병합해야 합니다.
4.1 99와 같이 미리 선물 개수를 선택하면 매개변수에 99가 포함된 메시지를 보낼 수 있습니다.
4.2 선물을 여러 번 보내고 총 개수가 불확실한 경우 선물 20개마다 메시지를 보내거나(값 조정 가능) 1초 이내에 클릭할 수 있습니다. 예를 들어 99개의 선물을 연속으로 클릭하면 최적화 후 5개의 메시지만 보내면 됩니다.

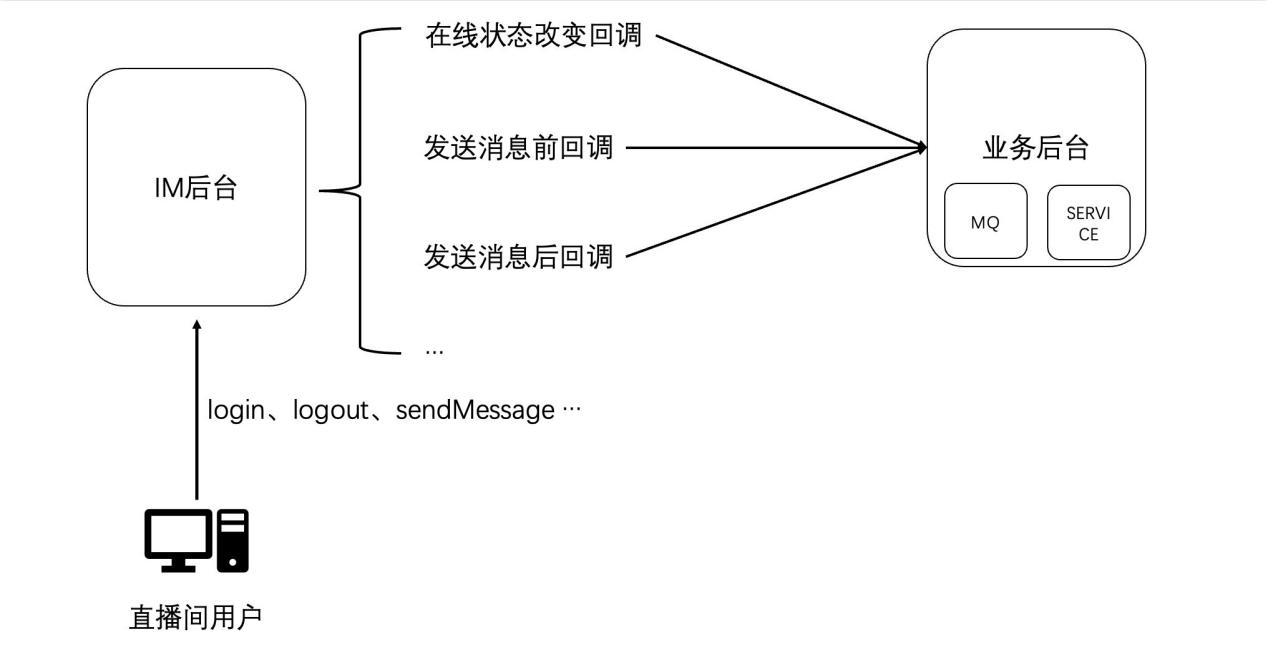
좋아요 메시지

1. 선물 메시지와 달리 좋아요 메시지는 청구되지 않으며 고객에게 직접 전송됩니다.
2. 서버에서 집계해야 하는 좋아요 메시지의 경우 클라이언트에서 트래픽 스로틀링이 수행된 후 클라이언트에서 좋아요가 집계되고 짧은 시간 동안의 좋아요 메시지가 하나로 병합됩니다. 비즈니스 서버는 메시지를 보내기 전에 콜백에서 좋아요 수를 얻습니다.
3. 계산할 필요가 없는 유사 메시지의 경우 2단계의 로직이 사용됩니다. 여기서 비즈니스 서버는 클라이언트에서 트래픽 스로틀링이 수행된 후 메시지를 보내고 메시지를 보내기 전에 콜백에서 카운트를 가져올 필요가 없습니다.

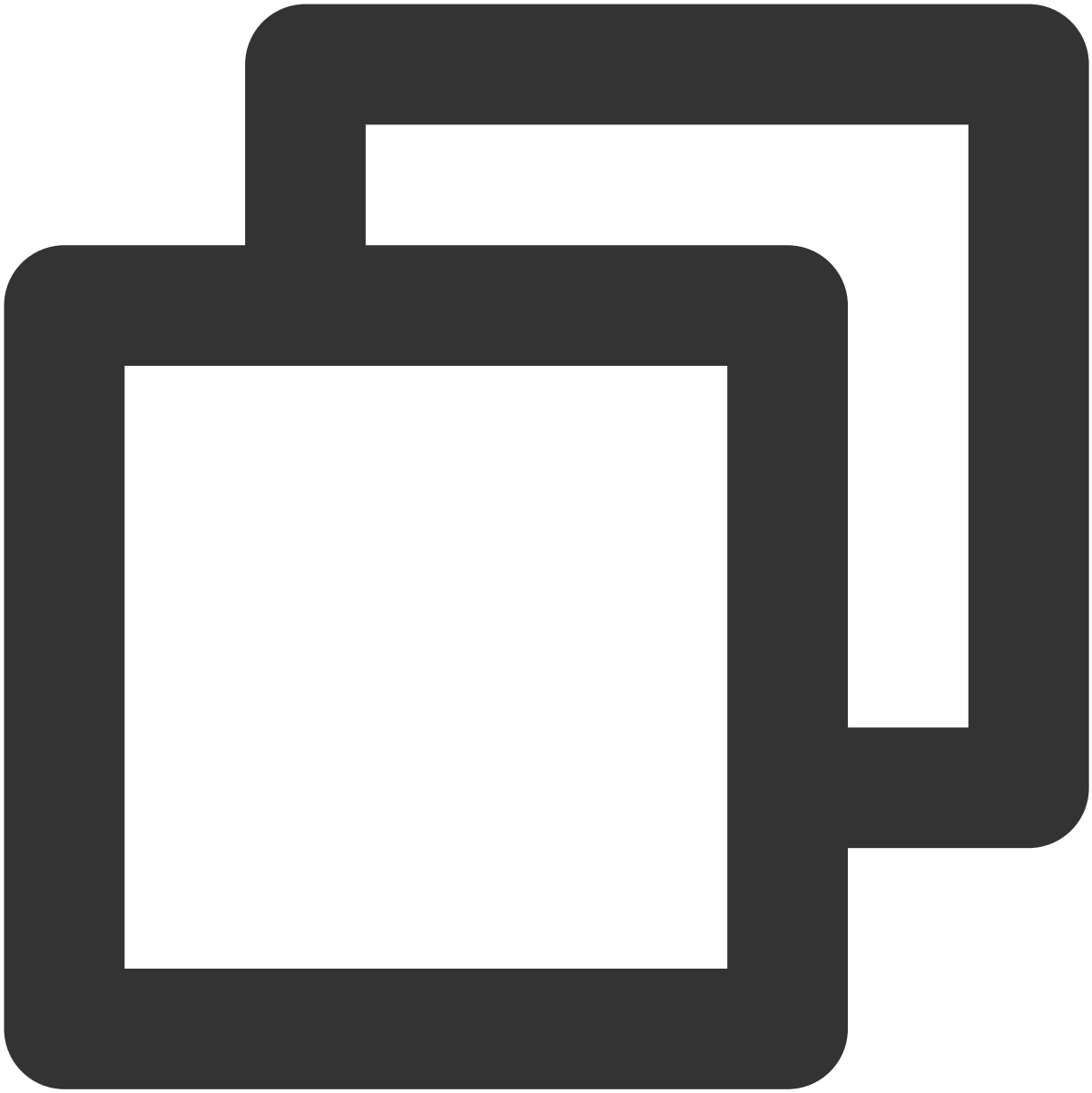
라이브 룸 사용자의 신분 및 레벨

사용자 레벨의 개념은 대부분의 라이브 룸에 적용됩니다. 다음을 기반으로 가중 사용자 레벨을 결정할 수 있습니다.
1. 라이브 룸 사용자의 온라인 지속 시간
2. 라이브 룸 사용자가 전송 완료한 일반 메시지 수
3. 라이브룸 사용자가 보낸 좋아요 및 선물 수
4. 사용자의 라이브 룸의 회원 권한 보유 여부
5. ...
온라인 지속 시간 및 메시지 수의 경우 준비 지침에 따라 구성할 수 있는 IM 콜백을 사용해야 합니다. 콘솔의 콜백은 다음과 같습니다.

정보 수집을 위한 순서도는 아래와 같습니다.



메시지 전송 후 콜백의 예시 데이터:

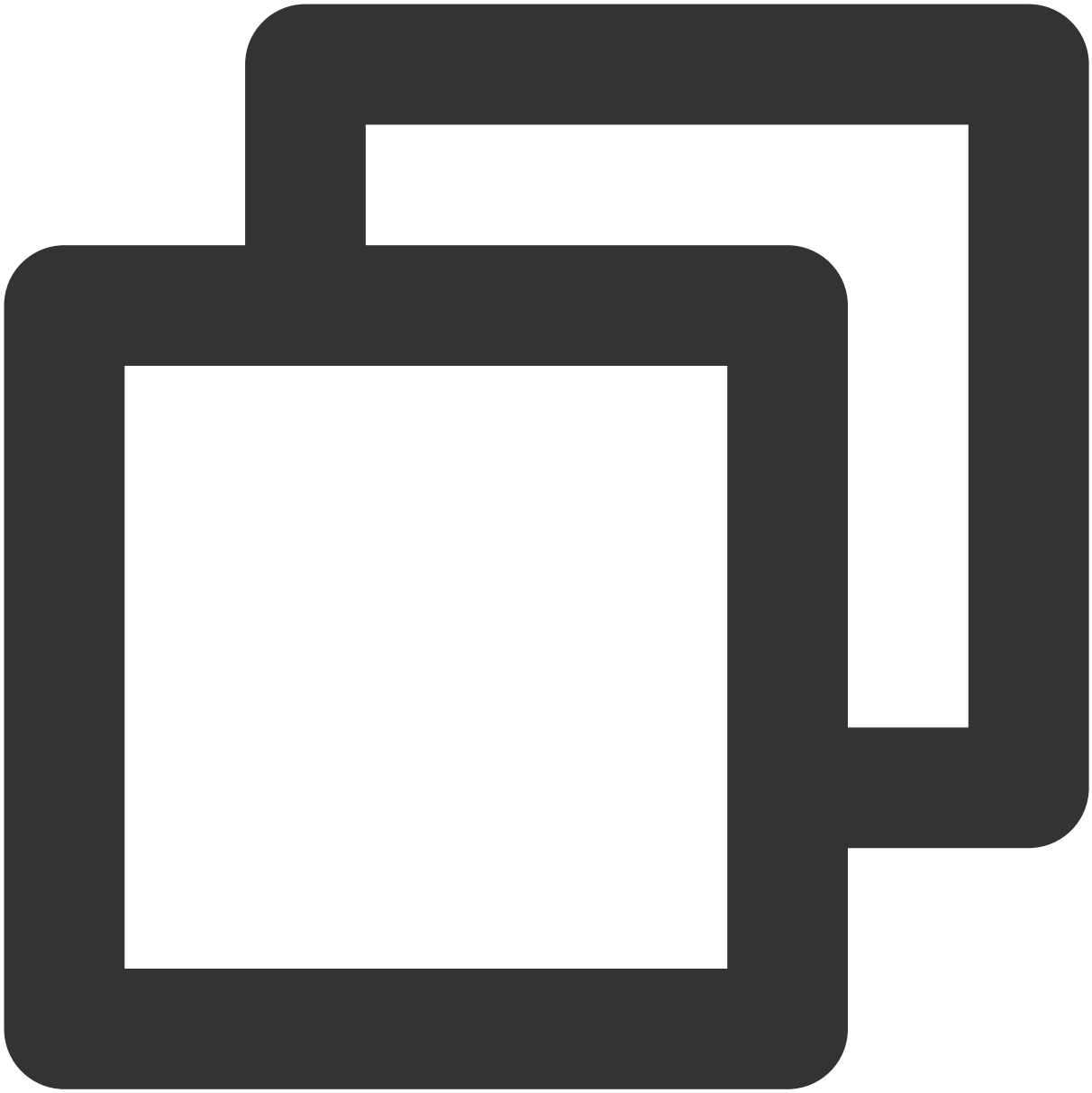


```
{
    "CallbackCommand": "Group.CallbackAfterSendMsg", // 콜백 명령
    "GroupId": "@TGS#2J4SZEAEL", // 그룹 ID
    "Type": "Public", // 그룹 유형
    "From_Account": "jared", // 발신자
    "Operator_Account":"admin", // 요청 개시자
    "Random": 123456, // 랜덤 숫자
    "MsgSeq": 123, // 메시지의 시퀀스 번호
    "MsgTime": 1490686222, // 메시지 시간
    "OnlineOnlyFlag": 1, // 값은 온라인 메시지인 경우 1이고 그렇지 않은 경우 0(기본값)입니다
    "MsgBody": [ // 메시지 본문, 자세한 내용은 TIMMessage 메시지 객체 참고
```

```
            {
                "MsgType": "TIMTextElem", // 텍스트
                "MsgContent": {
                    "Text": "red packet"
                }
            }
        ],
        "CloudCustomData": "your cloud custom data"
    }
```

MsgBody의 메시지 유형에 따라 일반, 좋아요, 선물 메시지를 식별할 수 있습니다. 모든 필드에 대한 자세한 내용은 Callback After Sending a Group Message를 참고하십시오.

사용자 온라인 상태 변경에 대한 콜백 데이터 예시:



```
{
    "CallbackCommand": "State.StateChange",
    "EventTime": 1629883332497,
    "Info": {
        "Action": "Login",
        "To_Account": "testuser316",
        "Reason": "Register"
    },
    "KickedDevice": [
        {
            "Platform": "Windows"
```

```
            },
            {
                "Platform": "Android"
            }
        ]
    }
```

온라인 지속 시간을 계산하기 위해 Info의 필드를 기반으로 사용자 온라인 상태를 식별할 수 있습니다. 모든 필드에 대한 자세한 내용은 State Change Callbacks를 참고하십시오.

설명：

또한, 라이브 룸 인기도를 표시하여 라이브 룸 추천의 기준으로 활용하는 경우가 많습니다. 사용자 수준과 유사하게 결정되며 IM의 콜백 시스템을 통해 구현할 수 있습니다.

라이브 룸의 메시지 기록

오디오/비디오 그룹(AVChatRoom)에 대한 메시지 기록은 기본적으로 저장되지 않습니다. 새로운 사용자가 라이브 룸에 입장하면 해당 사용자는 입장 후 보낸 메시지만 볼 수 있습니다. 사용자 경험을 최적화하기 위해 아래와 같이 가져올 수 있는 메시지 기록 수를 구성할 수 있습니다.

설명：

이 기능은 플래그십 에디션에서만 사용할 수 있으며 지난 24시간 동안 최대 20개의 메시지 기록을 가져올 수 있습니다.

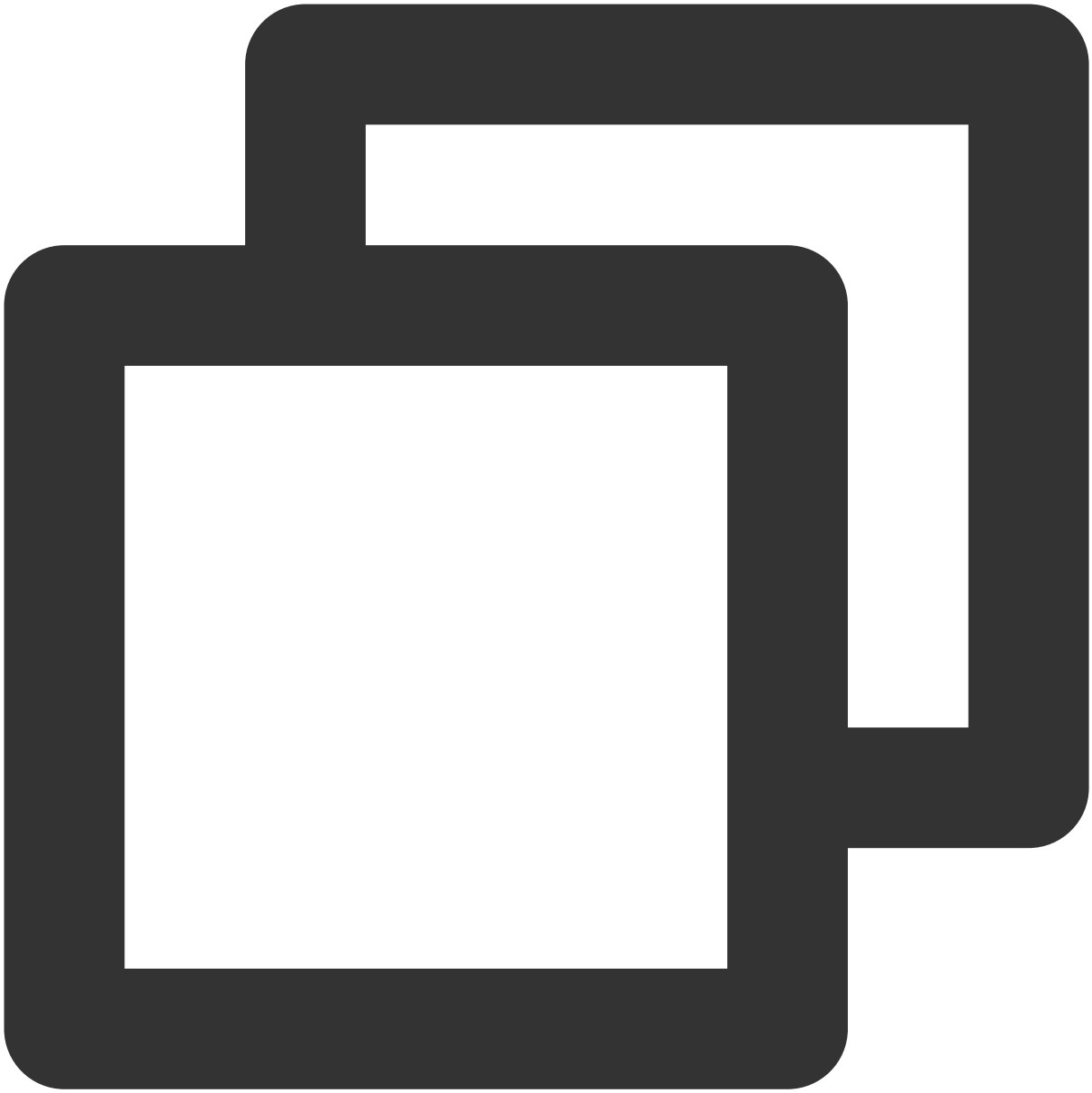
오디오/비디오 그룹의 메시지 기록은 다른 그룹과 동일한 방식으로 가져옵니다. 샘플 코드:

Android

iOS&Mac

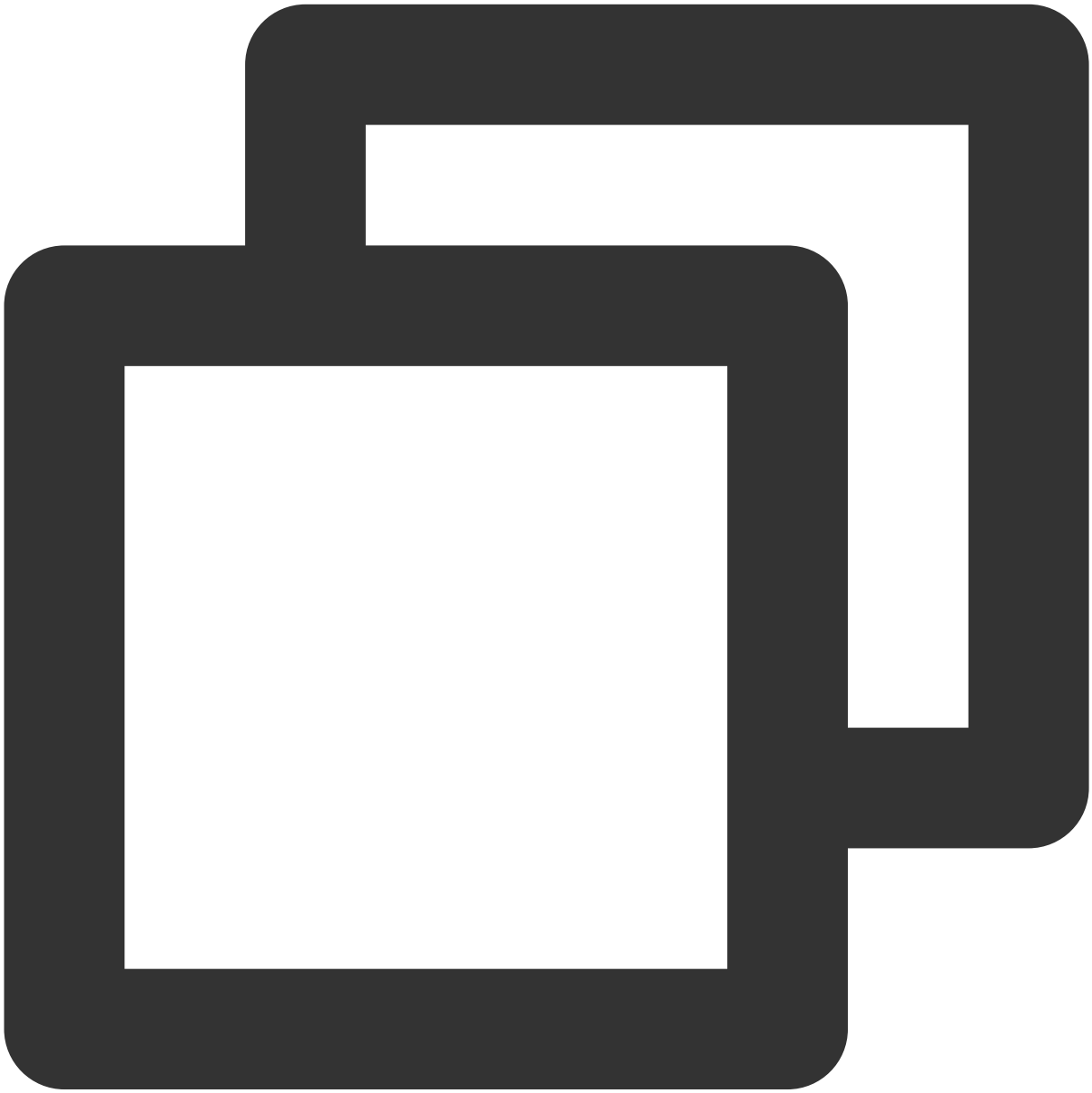
Flutter

Web



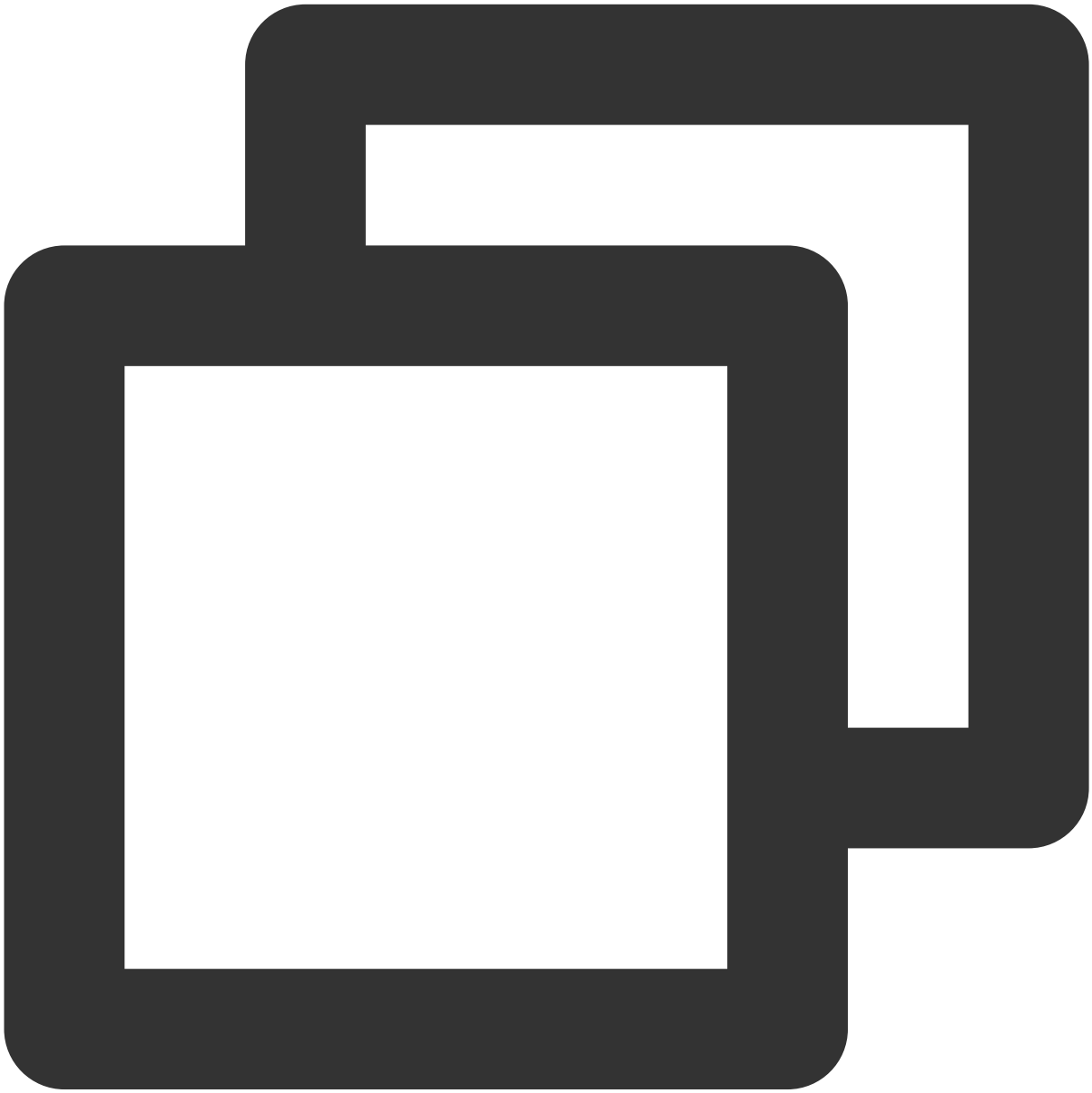
```
V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGetType(V2TIMMessageListGetOption.V2TIM_GET_CLOUD_OLDER_MSG); // 더 오래된
option.setGetTimeBegin(1640966400);          // 2022-01-01 00:00:00부터 시작
option.setGetTimePeriod(1 * 24 * 60 * 60);   // 하루 전체의 메시지 풀링
option.setCount(Integer.MAX_VALUE);          // 시간 범위 내의 모든 메시지 반환
option.setGroupID(#you group id#);           // 그룹 메시지 풀링
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCallba
    @Override
    public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
        Log.i("imsdk", "success");
    }
```

```
    @Override
    public void onError(int code, String desc) {
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```



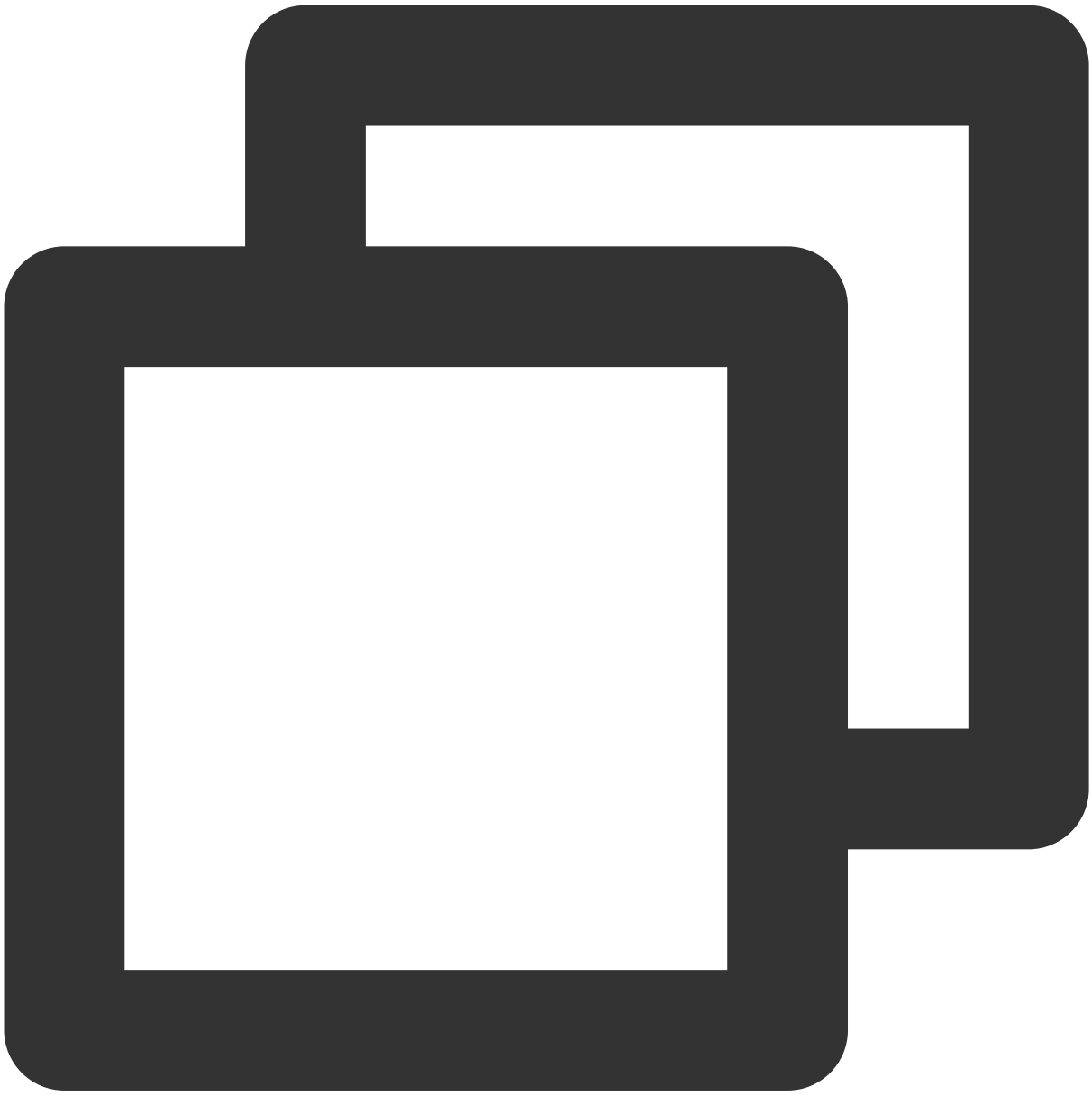
```
// 일대일 메시지 기록 풀링
// 첫 번째 풀링에 대해 lastMsg를 nil로 설정
// lastMsg는 두 번째 풀링에 대해 반환된 메시지 목록의 마지막 메시지일 수 있음
[V2TIMManager.sharedInstance getC2CHistoryMessageList:#your user id# count:20 lastM
```

```
    // 다음 풀링을 위해 lastMsg 기록
    V2TIMMessage *lastMsg = msgs.lastObject;
    NSLog(@"success, %@", msgs);
} fail:^(int code, NSString *desc) {
    NSLog(@"fail, %d, %@", code, desc);
}];
```



```
// 일대일 메시지 기록 풀링
// 첫 번째 풀링에 대해 lastMsgID를 null로 설정
// lastMsgID는 두 번째 풀링에 대해 반환된 메시지 목록의 마지막 메시지 id일 수 있음
TencentImSDKPlugin.v2TIMManager.getMessageManager().getC2CHistoryMessageList(
```

```
        userID: "userId",
        count: 10,
        lastMsgID: null,
);
```



```
// 대화가 열릴 때 처음 메시지 목록 풀링
let promise = tim.getMessageList({conversationID: 'C2Ctest', count: 15});
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // 메시지 리스트.
  const nextReqMessageID = imResponse.data.nextReqMessageID; // 이 매개변수는 다음 페이
  const isCompleted = imResponse.data.isCompleted; // 모든 메시지가 풀링되었는지 여부를
```

```
});
// 메시지 리스트를 풀 다운하여 더 많은 메시지 보기
let promise = tim.getMessageList({conversationID: 'C2Ctest', nextReqMessageID, coun
promise.then(function(imResponse) {
  const messageList = imResponse.data.messageList; // 메시지 리스트.
  const nextReqMessageID = imResponse.data.nextReqMessageID; // 이 매개변수는 다음 페이
  const isCompleted = imResponse.data.isCompleted; // 모든 메시지가 풀링되었는지 여부를
});
```

다른 SDK의 메시지 기록 가져오기 예시 코드

라이브 룸의 온라인 사용자 수

라이브 룸의 온라인 사용자 수를 표시하는 것은 일반적인 기능입니다. 각각 장단점이 있는 두 가지 구현 스키마가 있습니다.

1. 클라이언트 SDK에서 제공하는 getGroupOnlineMemberCount API를 통해 스케쥴된 풀링 방식으로 그룹의 온라인 사용자 수를 가져옵니다.

2. 백엔드에서 그룹에 가입하거나 탈퇴하는 콜백을 기반으로 Sending System Messages in a Group 또는 Sending Ordinary Messages in a Group과 같은 서버 API를 통해 모든 그룹 구성원에게 데이터를 전달합니다.

라이브 룸이 하나뿐이라면 클라이언트 SDK의 API를 통해 온라인 사용자 수를 풀링하기에 충분합니다. 온라인 사용자 수를 표시하기 위해 많은 노출 위치가 필요한 라이브 룸이 여러 개 있는 경우 두 번째 스키마를 권장합니다.

설명：

서버는 예를 들어 5초마다 한 번씩 스케쥴된 방식으로 온라인 사용자 수 통계 메시지를 클라이언트에 보낼 수 있습니다. 그러나 라이브 룸의 온라인 사용자 수가 급격히 변경되지 않으면 추가 네트워크 오버헤드가 발생합니다. 변경률을 모니터링하여 숫자를 업데이트하는 것이 좋습니다.

필요에 따라 라이브 룸의 온라인 사용자 수의 정확성과 실시간성의 우선순위를 결정할 수 있습니다.

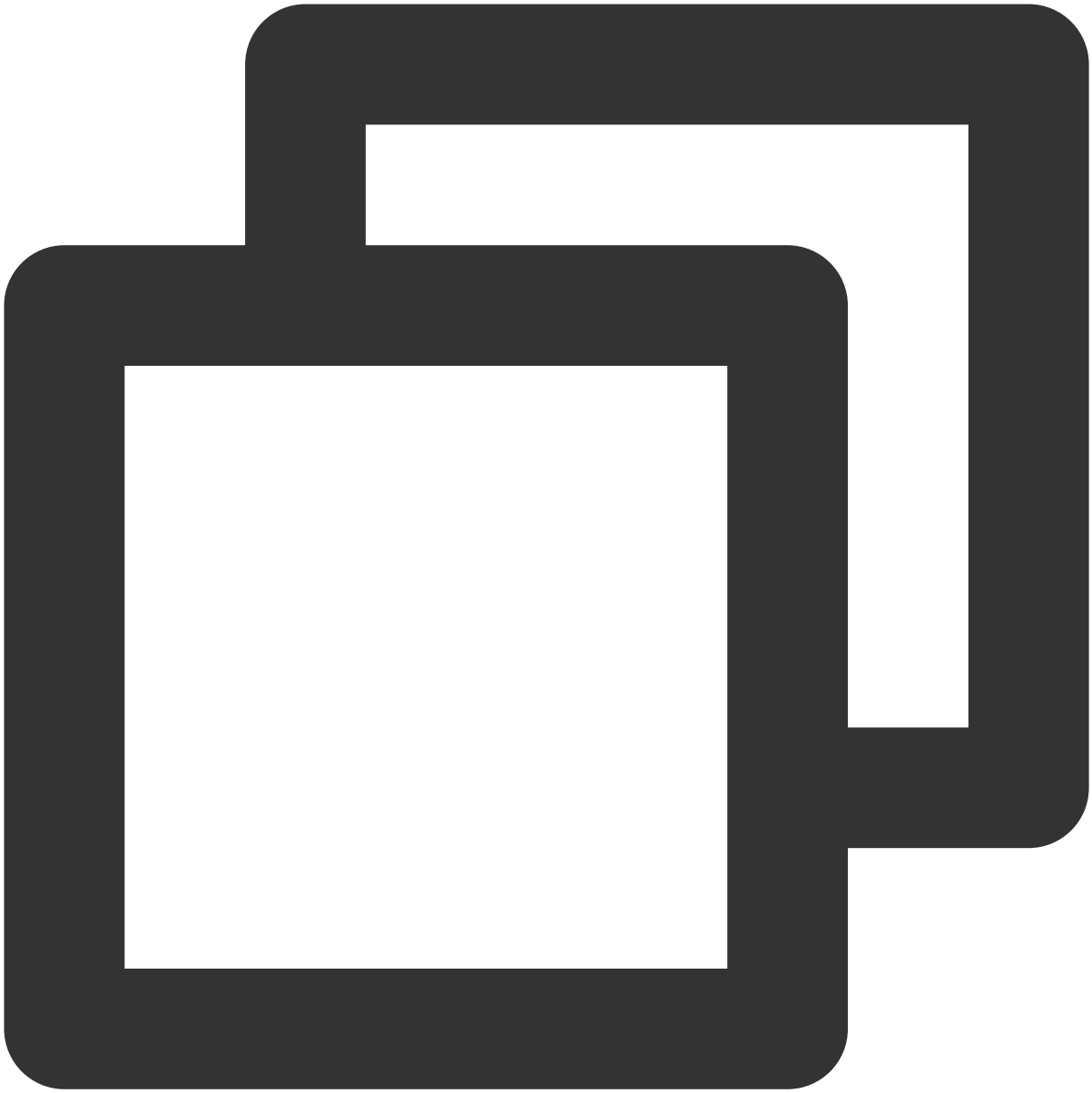
라이브 룸의 온라인 사용자 수를 가져오는 코드는 다음과 같습니다.

Android

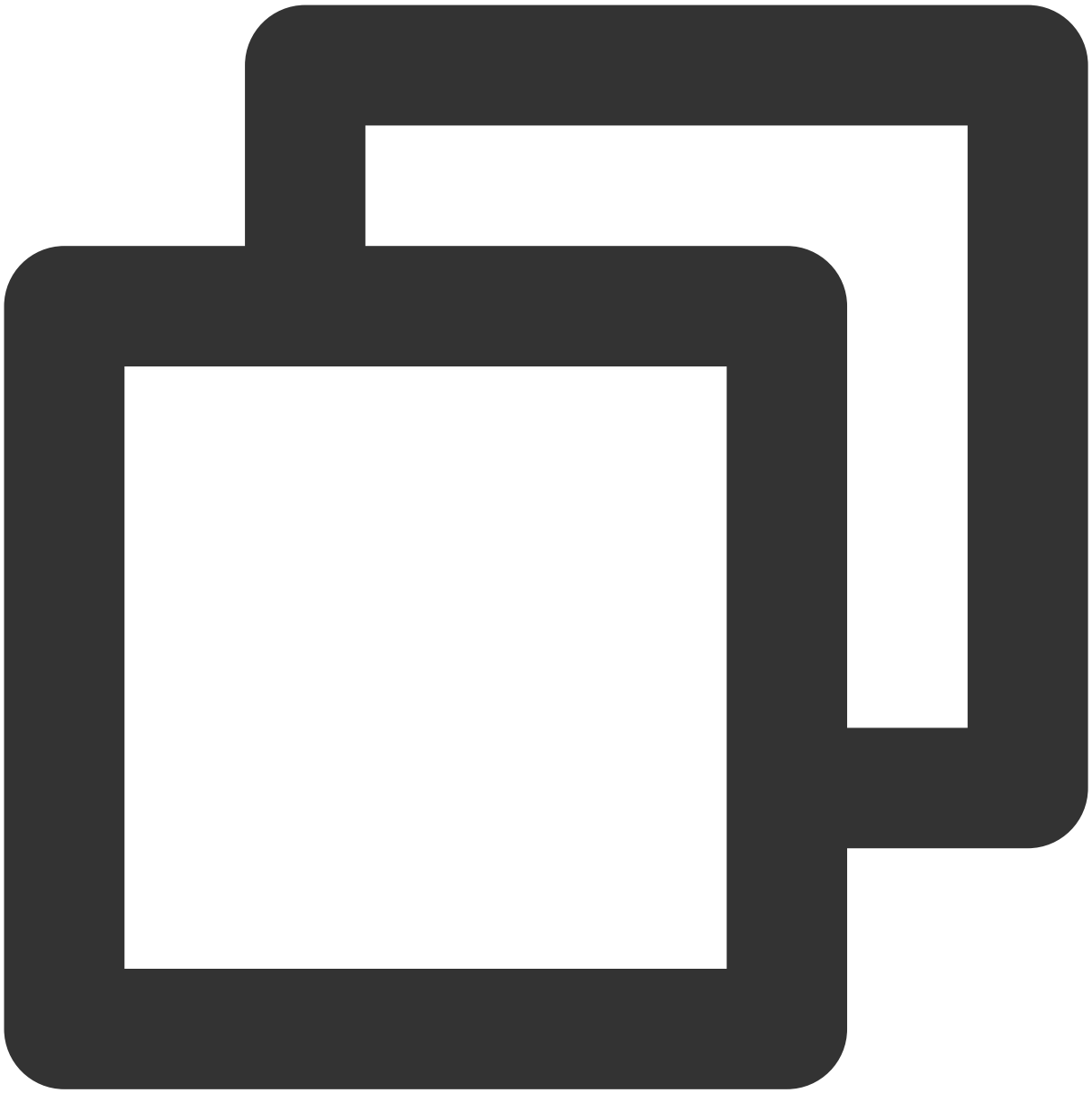
iOS&Mac

Flutter

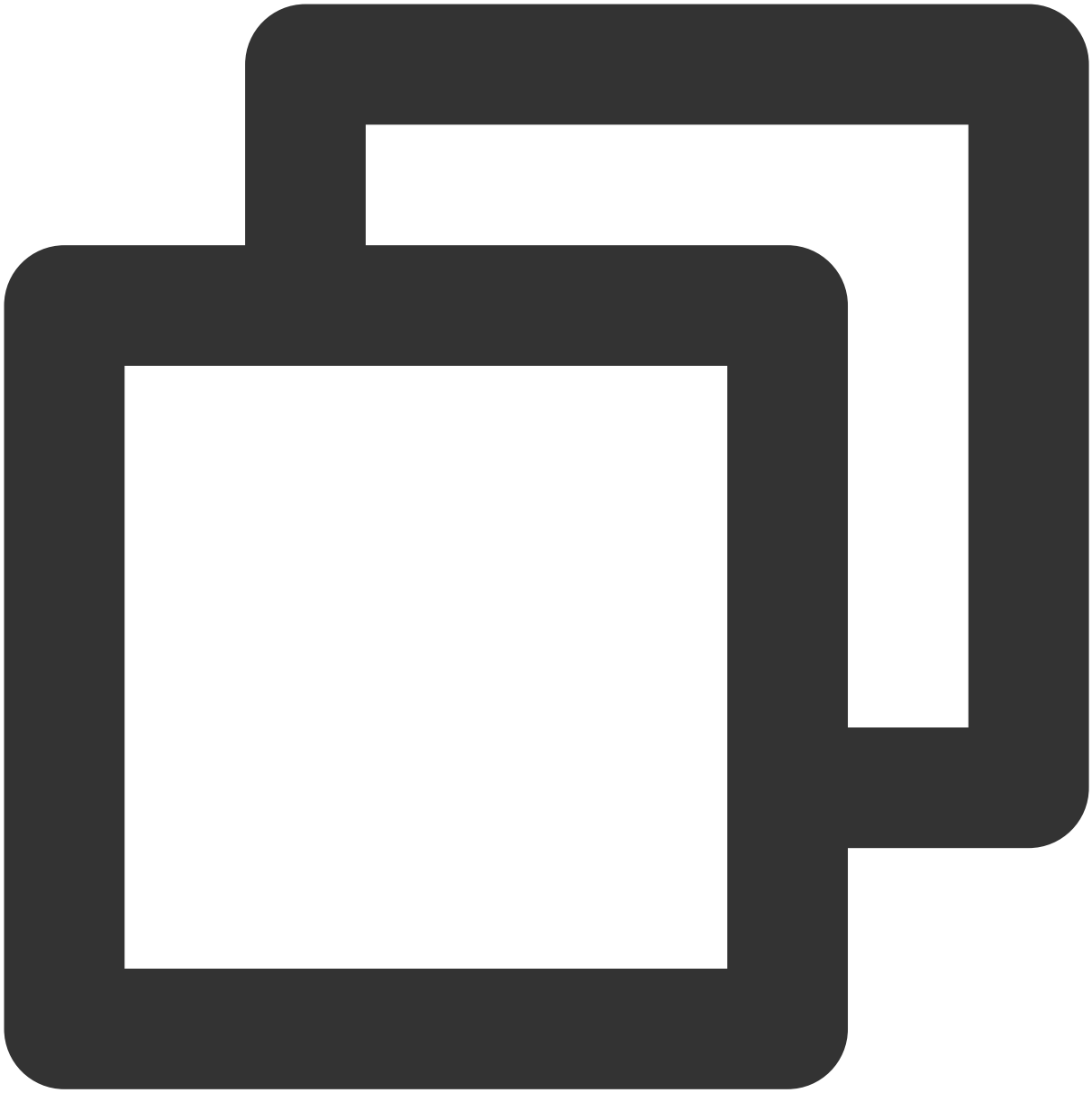
Web



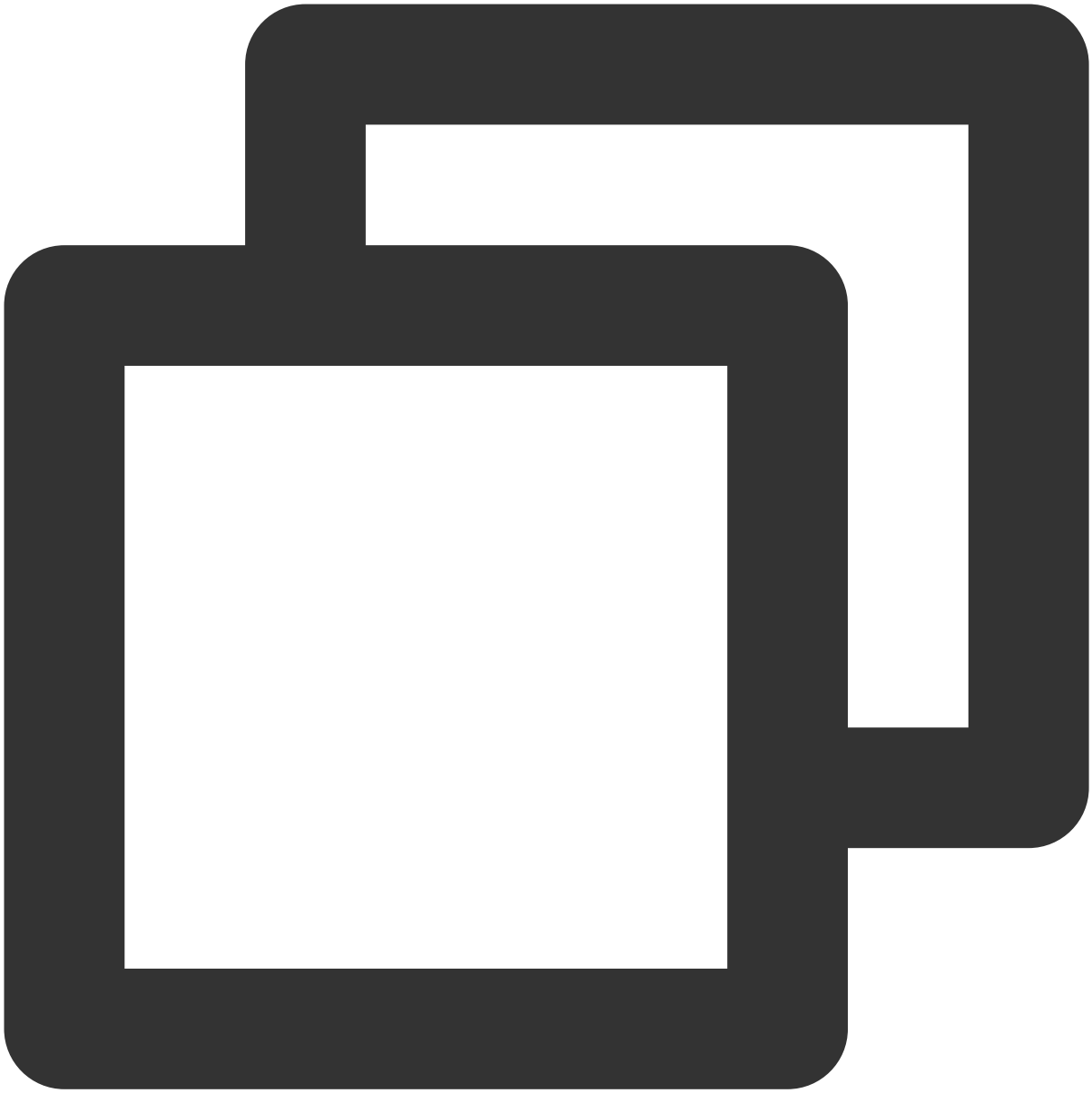
```
2TIMManager.getGroupManager().getGroupOnlineMemberCount("group_avchatroom", new V2T
  @Override
  public void onSuccess(Integer integer) {
      // 오디오/비디오 그룹(AVChatRoom)의 온라인 구성원 수 가져오기 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 오디오/비디오 그룹(AVChatRoom)의 온라인 구성원 수 가져오기 실패
  }
});
```



```
[[V2TIMManager sharedInstance] getGroupOnlineMemberCount:@"group_avchatroom" succ:^
    // 오디오/비디오 그룹(AVChatRoom)의 온라인 구성원 수 가져오기 성공
} fail:^(int code, NSString *desc) {
    // 오디오/비디오 그룹(AVChatRoom)의 온라인 구성원 수 가져오기 실패
}];
```



```
groupManager.getGroupOnlineMemberCount(groupID: '');
```



```
// 오디오/비디오 그룹의 온라인 사용자 수 가져오기(v2.8.0부터 지원)
let promise = tim.getGroupOnlineMemberCount('group1');
promise.then(function(imResponse) {
  console.log(imResponse.data.memberCount);
}).catch(function(imError) {
  console.warn('getGroupOnlineMemberCount error:', imError); // 오디오/비디오 그룹(AVC
});
```

라이브 룸에서 사용자 음소거

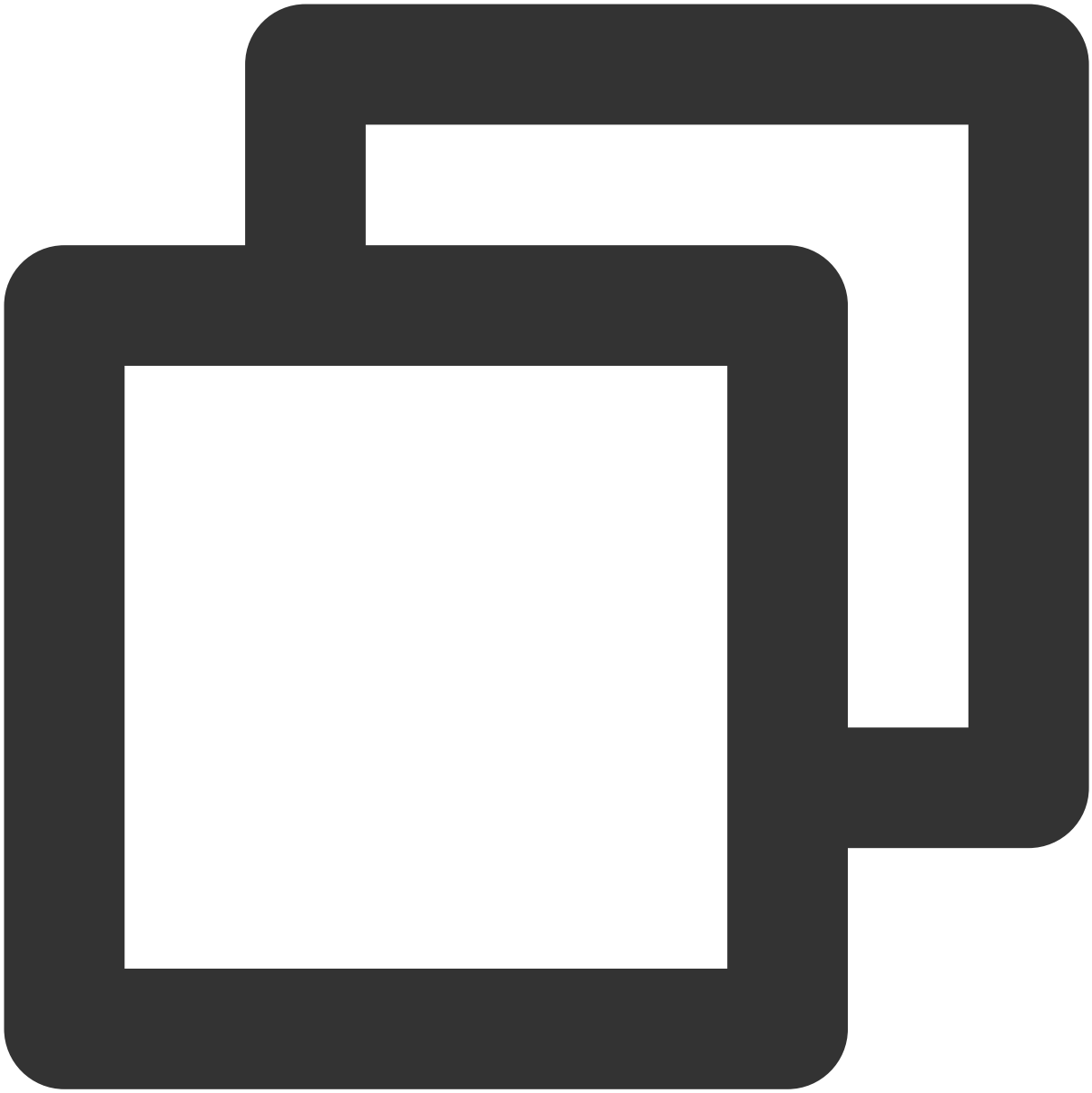
라이브 룸에서는 다양한 시나리오에서 모든 사용자 또는 지정된 사용자를 음소거할 수 있습니다.
일반적으로 서버 SDK는 음소거에 사용됩니다. 전체 음소거하려면 Modifying the Profile of a Group에 설명된 대로 그룹 속성의 MuteAllMember 필드를 설정합니다. 지정된 사용자를 음소거하려면 Bulk Muting and Unmuting의 지침에 따라 그룹 구성원 속성을 설정하십시오.
라이브 룸 관리자가 백엔드에서 그룹 음소거를 설정할 때 클라이언트는 콜백 이벤트를 수신한 후 타깃 사용자의 입력 상자를 disable 상태로 설정해야 합니다. 그렇지 않으면 사용자는 메시지를 보낼 때 메시지 전송 실패 프롬프트를 받게 됩니다. 사용자의 음소거가 해제되면 클라이언트도 입력 상자를 enable 상태로 설정해야 합니다.
클라이언트의 콜백 코드:

Android

iOS&Mac

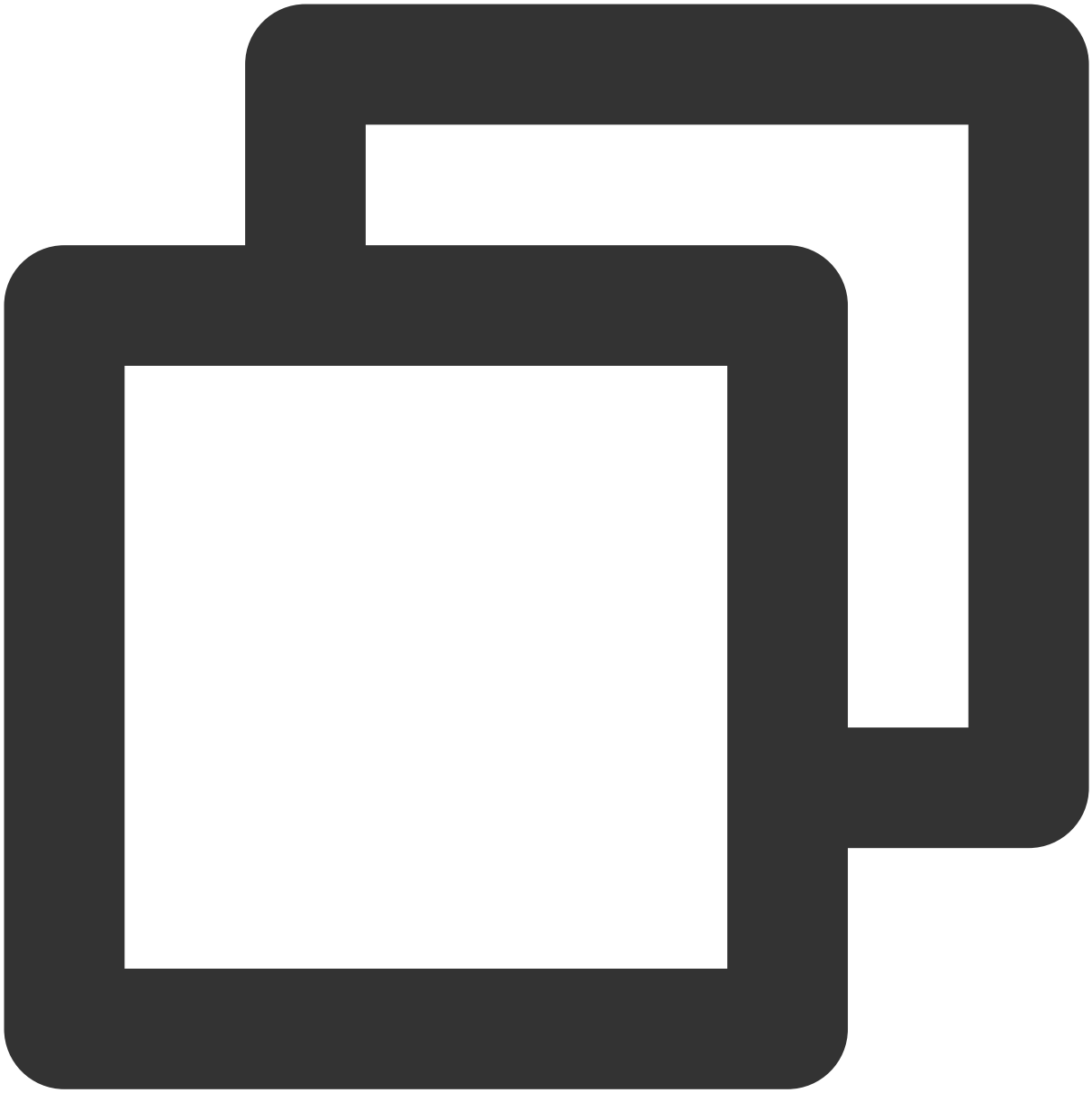
Flutter

Web



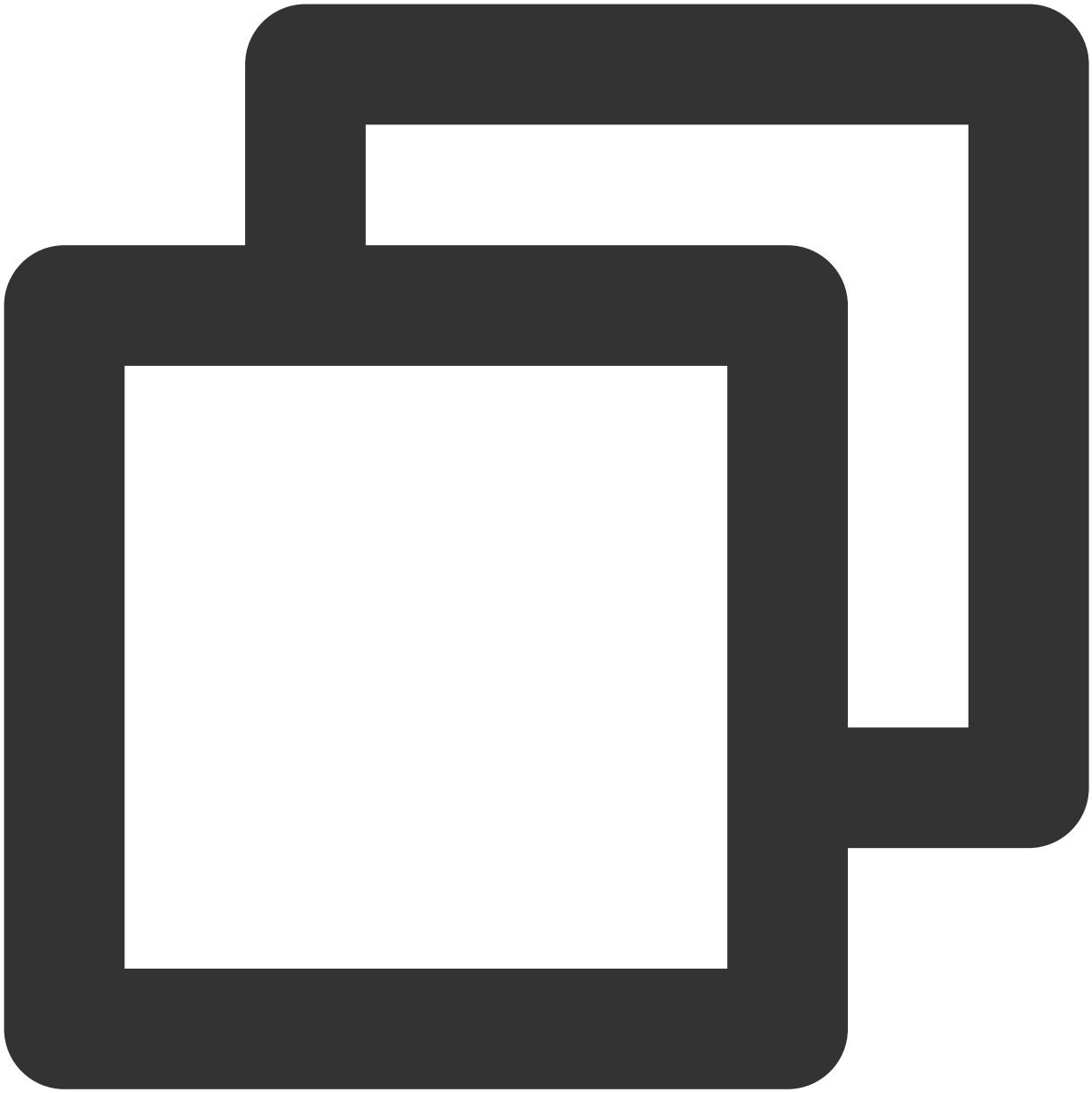
```
// 1분 동안 그룹 구성원 userB 음소거
V2TIMManager.getGroupManager().muteGroupMember("groupA", "userB", 60, new V2TIMCall
  @Override
  public void onSuccess() {
      // 그룹 구성원 음소거 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 그룹 구성원 음소거 실패
  }
```

```
});

// 모든 구성원 음소거
V2TIMGroupInfo info = new V2TIMGroupInfo();
info.setGroupID("groupA");
info.setAllMuted(true);
V2TIMManager.getGroupManager().setGroupInfo(info, new V2TIMCallback() {
  @Override
  public void onSuccess() {
      // 모든 구성원 음소거 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 모든 구성원 음소거 실패
  }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
  @Override
  public void onMemberInfoChanged(String groupID, List<V2TIMGroupMemberChangeInfo>
    // 그룹 구성원의 음소거 수신
    for (V2TIMGroupMemberChangeInfo memberChangeInfo : v2TIMGroupMemberChangeInfoLi
      // 음소거된 사용자의 ID
      String userID = memberChangeInfo.getUserID();
      // 음소거 지속 시간
      long muteTime = memberChangeInfo.getMuteTime();
    }
  }

  @Override
  public void onGroupInfoChanged(String groupID, List<V2TIMGroupChangeInfo> changeI
    // 모든 구성원 음소거 수신
    for (V2TIMGroupChangeInfo groupChangeInfo : changeInfos) {
      if (groupChangeInfo.getType() == V2TIMGroupChangeInfo.V2TIM_GROUP_INFO_CHANGE
        // 모든 구성원 음소거 여부
        boolean isMuteAll = groupChangeInfo.getBoolValue();
      }
    }
  }
});
```



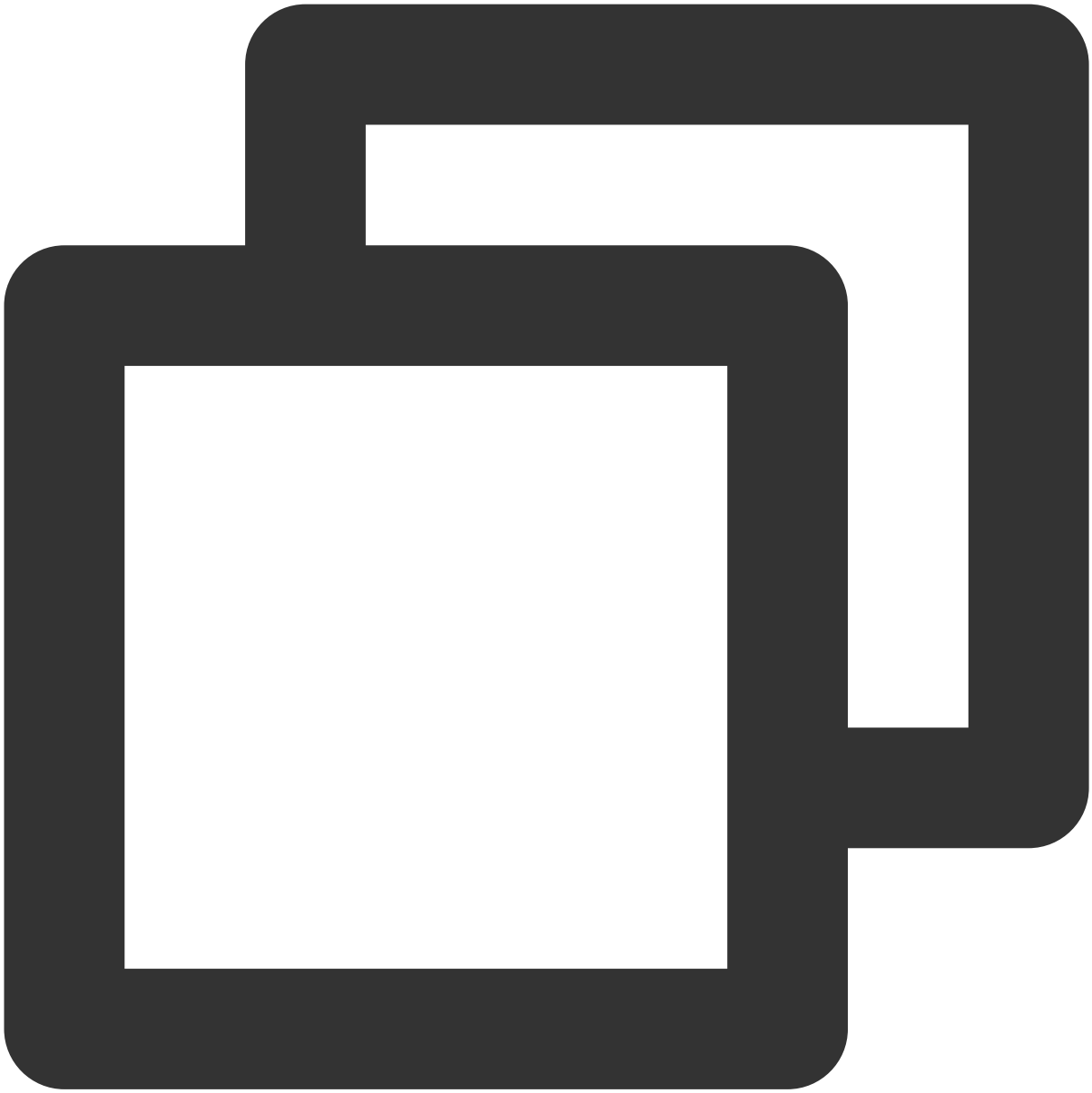
```
// 1분 동안 그룹 구성원 user1 음소거
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA" member:@"user1" muteTime:6
    // 그룹 구성원 음소거 성공
} fail:^(int code, NSString *desc) {
    // 그룹 구성원 음소거 실패
}];

// 모든 구성원 음소거
V2TIMGroupInfo *info = [[V2TIMGroupInfo alloc] init];
info.groupID = @"groupA";
info.allMuted = YES;
```

```
[[V2TIMManager sharedInstance] muteGroupMember:@"groupA" member:@"user1" muteTime:6
    // 모든 구성원 음소거 성공
} fail:^(int code, NSString *desc) {
    // 모든 구성원 음소거 실패
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberInfoChanged:(NSString *)groupID changeInfoList:(NSArray <V2TIMGroup
    // 그룹 구성원의 음소거 수신
    for (V2TIMGroupMemberChangeInfo *memberChangeInfo in changeInfoList) {
      // 음소거된 사용자의 ID
      NSString *userID = memberChangeInfo.userID;
      // 음소거 지속 시간
        uint32_t muteTime = memberChangeInfo.muteTime;
    }
}

- (void)onGroupInfoChanged:(NSString *)groupID changeInfoList:(NSArray <V2TIMGroupC
    // 모든 구성원 음소거 수신
    for (V2TIMGroupChangeInfo groupChangeInfo in changeInfoList) {
      if (groupChangeInfo.type == V2TIM_GROUP_INFO_CHANGE_TYPE_SHUT_UP_ALL) {
        // 모든 구성원 음소거 여부
        BOOL isMuteAll = groupChangeInfo.boolValue;
      }
    }
}
```



```
// 10분 동안 그룹 구성원 userB 음소거
groupManager.muteGroupMember(groupID: '',userID: 'userB',seconds: 10);

// 모든 구성원 음소거
groupManager.setGroupInfo(info: V2TimGroupInfo(isAllMuted: true,groupID: '',groupTy

TencentImSDKPlugin.v2TIMManager.addGroupListener(listener: V2TimGroupListener(onMem
    //그룹 구성원 정보 변경
  },
  onGroupInfoChanged: (groupID,info){
    // 그룹 프로필 수정
```

```
  }
  ));
```



```
tim.setGroupMemberMuteTime(options);
let promise = tim.setGroupMemberMuteTime({
  groupID: 'group1',
  userID: 'user1',
  muteTime: 600 // 10분 동안 사용자 음소거; 0으로 설정되면 사용자 음소거 해제
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 새 그룹 프로필
```

```
  console.log(imResponse.data.member); // 새 그룹 구성원 프로필
}).catch(function(imError) {
  console.warn('setGroupMemberMuteTime error:', imError); // 음소거 실패 관련 오류 정보
});
// 주제 내 그룹 구성원 음소거 지속 시간 설정
let promise = tim.setGroupMemberMuteTime({
  groupID: 'topicID',
  userID: 'user1',
  muteTime: 600 // 10분 동안 사용자 음소거; 0으로 설정되면 사용자 음소거 해제
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 새 그룹 프로필
  console.log(imResponse.data.member); // 새 그룹 구성원 프로필
}).catch(function(imError) {
  console.warn('setGroupMemberMuteTime error:', imError); // 음소거 실패 관련 오류 정보
});
// v2.6.2부터 SDK는 모두 음소거 및 음소거 해제를 지원합니다. 현재 그룹 내 모든 구성원이 음소거
let promise = tim.updateGroupProfile({
  groupID: 'group1',
  muteAllMembers: true, // true: 전원 음소거, false: 전원 음소거 해제
});
promise.then(function(imResponse) {
  console.log(imResponse.data.group) // 수정 완료 후 세부 그룹 프로필
}).catch(function(imError) {
  console.warn('updateGroupProfile error:', imError); // 그룹 프로필 수정 실패 정보
});
```

다른 SDK의 그룹 수신 예시 코드

그룹 구성원 음소거 상태의 변경 사항은 기본적으로 클라이언트에 전달되지 않으며 콘솔에서 구성해야 합니다.

설명：

Client SDK는 현재 라이브 룸에서 사용자 음소거를 지원하지 않습니다. 해당 서버 API를 사용하여 Banning Group Members 및 Unbanning Group Members할 수 있습니다.

라이브 룸 차단

개발자는 서버의 Banning Group Members API를 통해 라이브 룸의 구성원을 강제 퇴장할 수 있으며, 강제 퇴장된 구성원이 그룹에 다시 들어올 수 없도록 일정 기간 동안 차단할 수 있습니다.
콘솔 구성 항목은 다음과 같습니다.

주의 :

** 클라이언트 SDK 6.6.X 이상 버전, Flutter SDK 4.1.1 이상 버전에서는 라이브 룸 구성원 내보내기 API를 사용하여 차단 기능을 구현할 수 있습니다.**

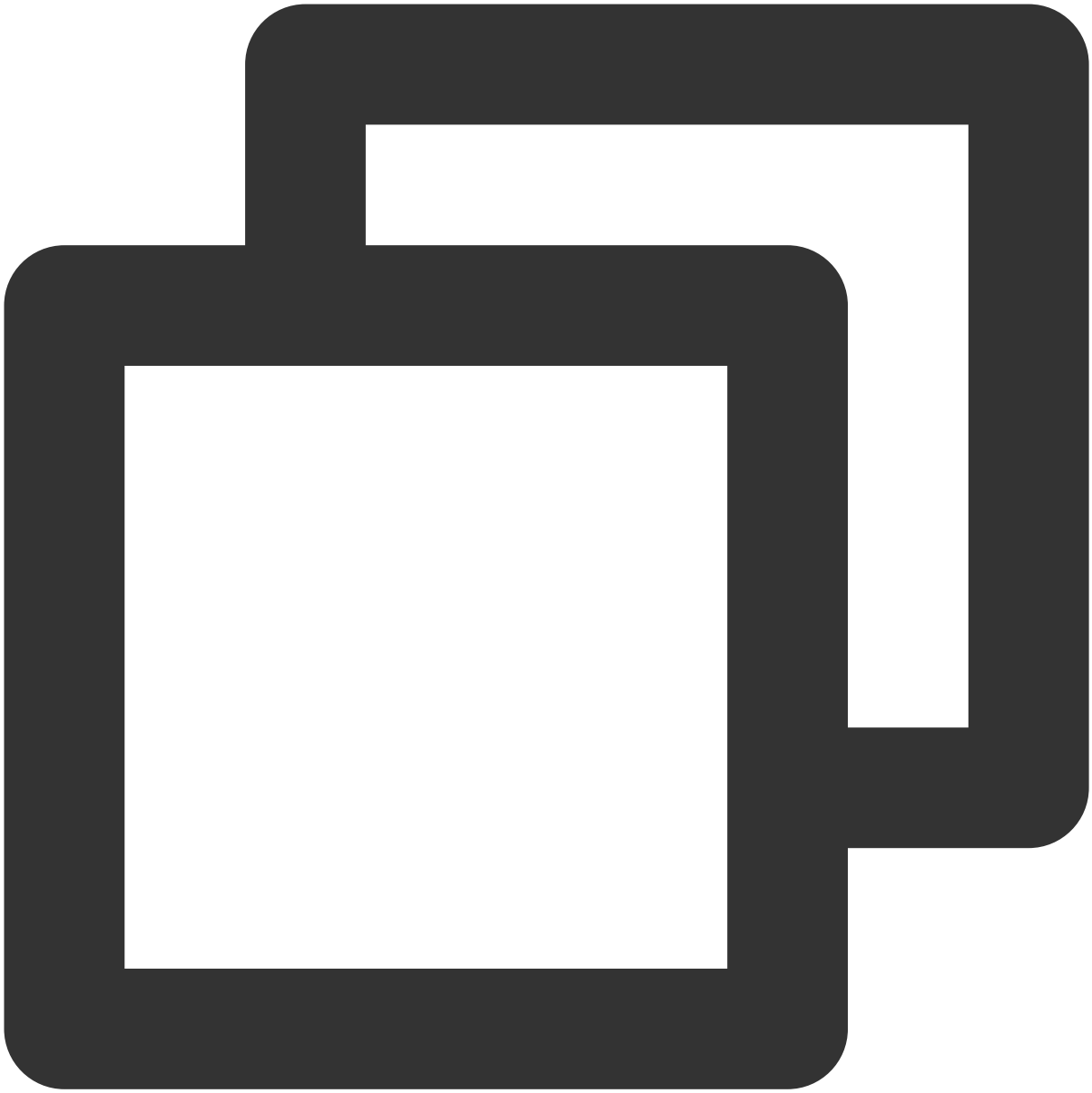
샘플 코드:

Android

iOS&Mac

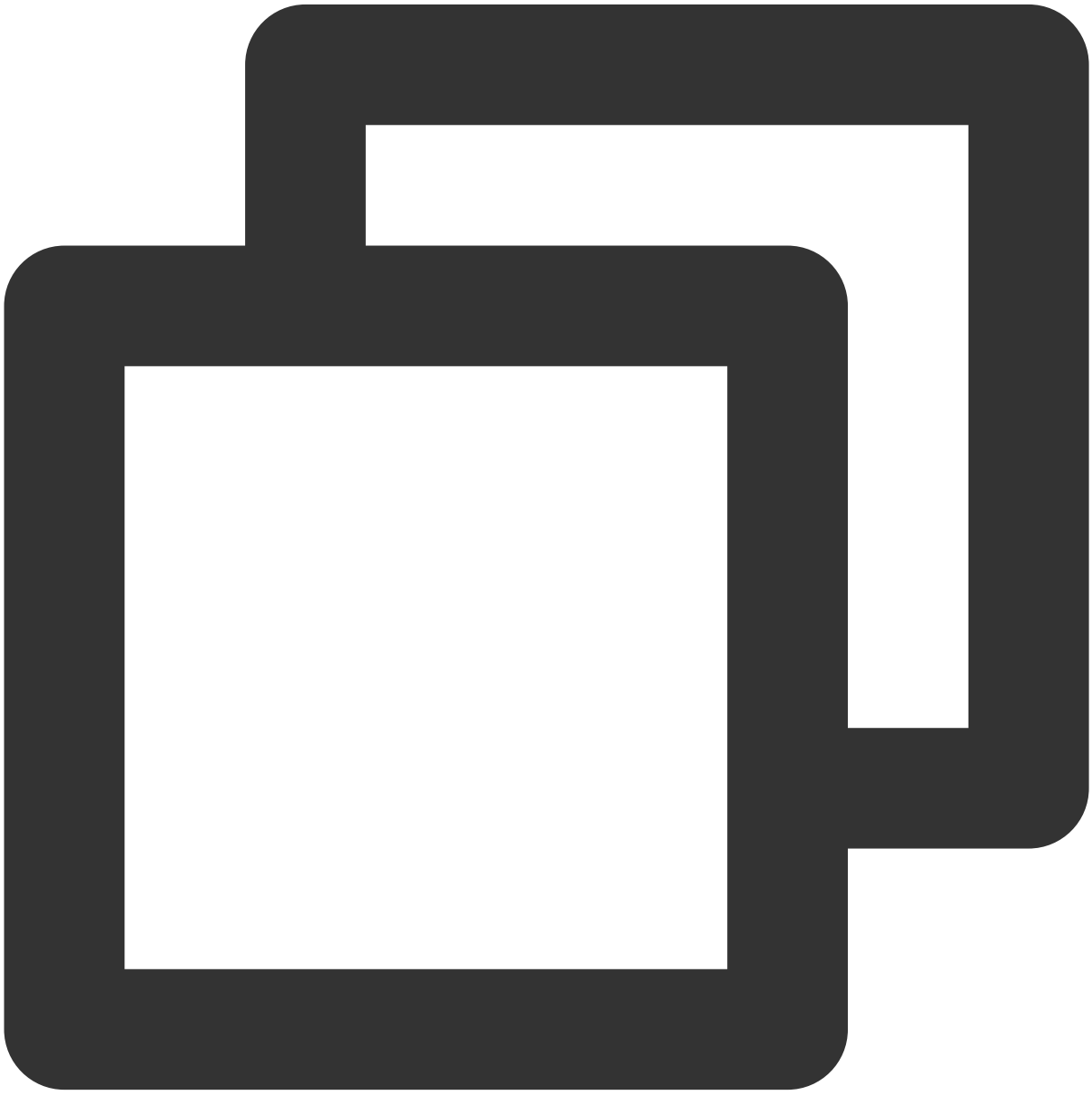
Flutter

Web

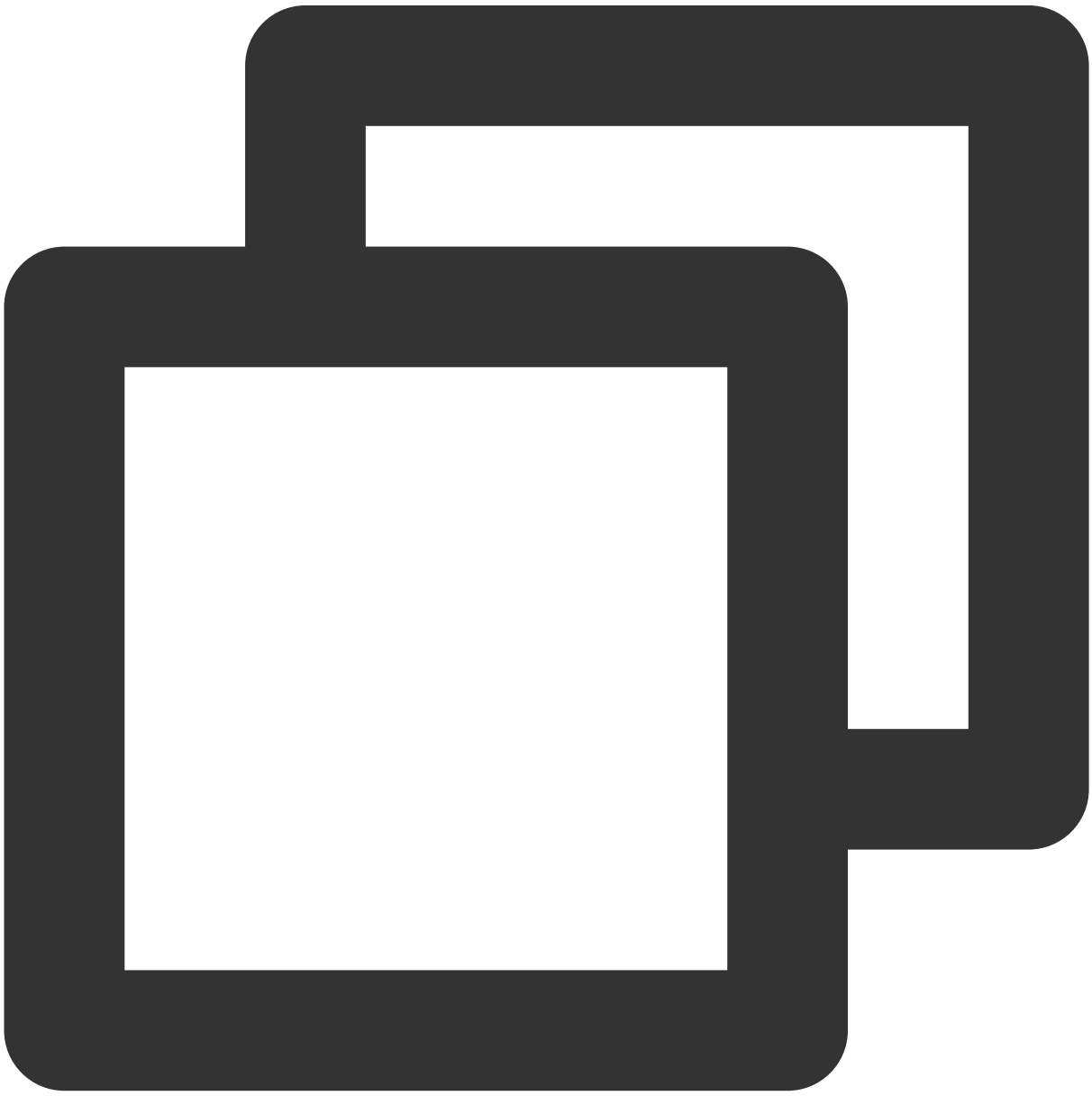


```
List<String> userIDList = new ArrayList<>();
userIDList.add("userB");
V2TIMManager.getGroupManager().kickGroupMember("groupA", userIDList, "", new V2TIMV
  @Override
  public void onSuccess(List<V2TIMGroupMemberOperationResult> v2TIMGroupMemberOpera
      // 구성원 제거 성공
  }

  @Override
  public void onError(int code, String desc) {
      // 구성원 제거 실패
```

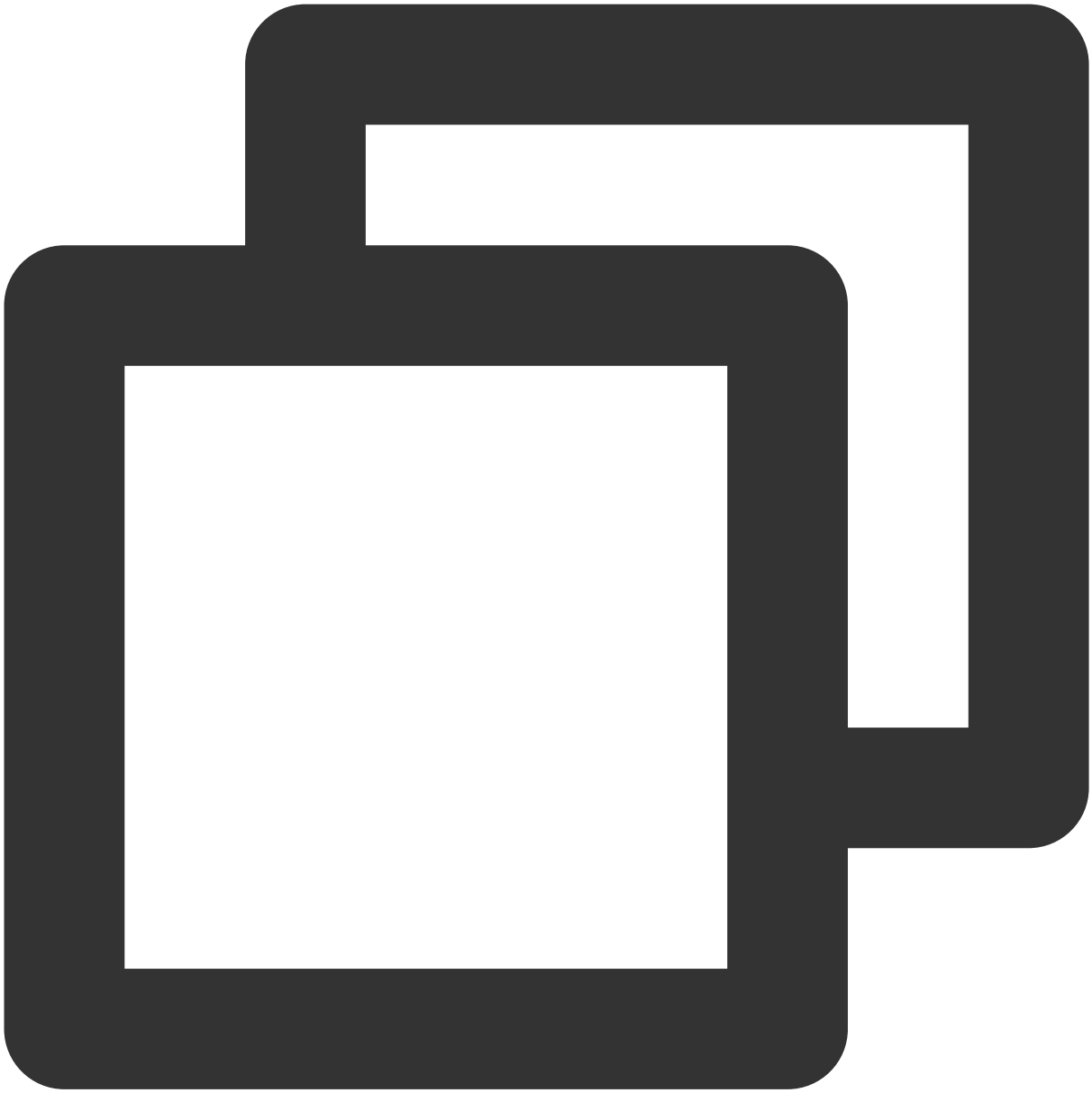
```
    }
});

V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
    @Override
    public void onMemberKicked(String groupID, V2TIMGroupMemberInfo opUser,
    List<V2TIMGroupMemberInfo> memberList) {
        // 그룹 구성원 제거됨
    }
});
```



```
[[V2TIMManager sharedInstance] kickGroupMember:@"groupA" memberList:@[@"user1"] rea
    // 구성원 제거 성공
} fail:^(int code, NSString *desc) {
    // 구성원 제거 실패
}];

[[V2TIMManager sharedInstance] addGroupListener:self];
- (void)onMemberKicked:(NSString *)groupID opUser:(V2TIMGroupMemberInfo *)opUser me
    // 그룹 구성원 제거됨
}
```



```
groupManager.kickGroupMember(groupID: '',memberList: []);
```

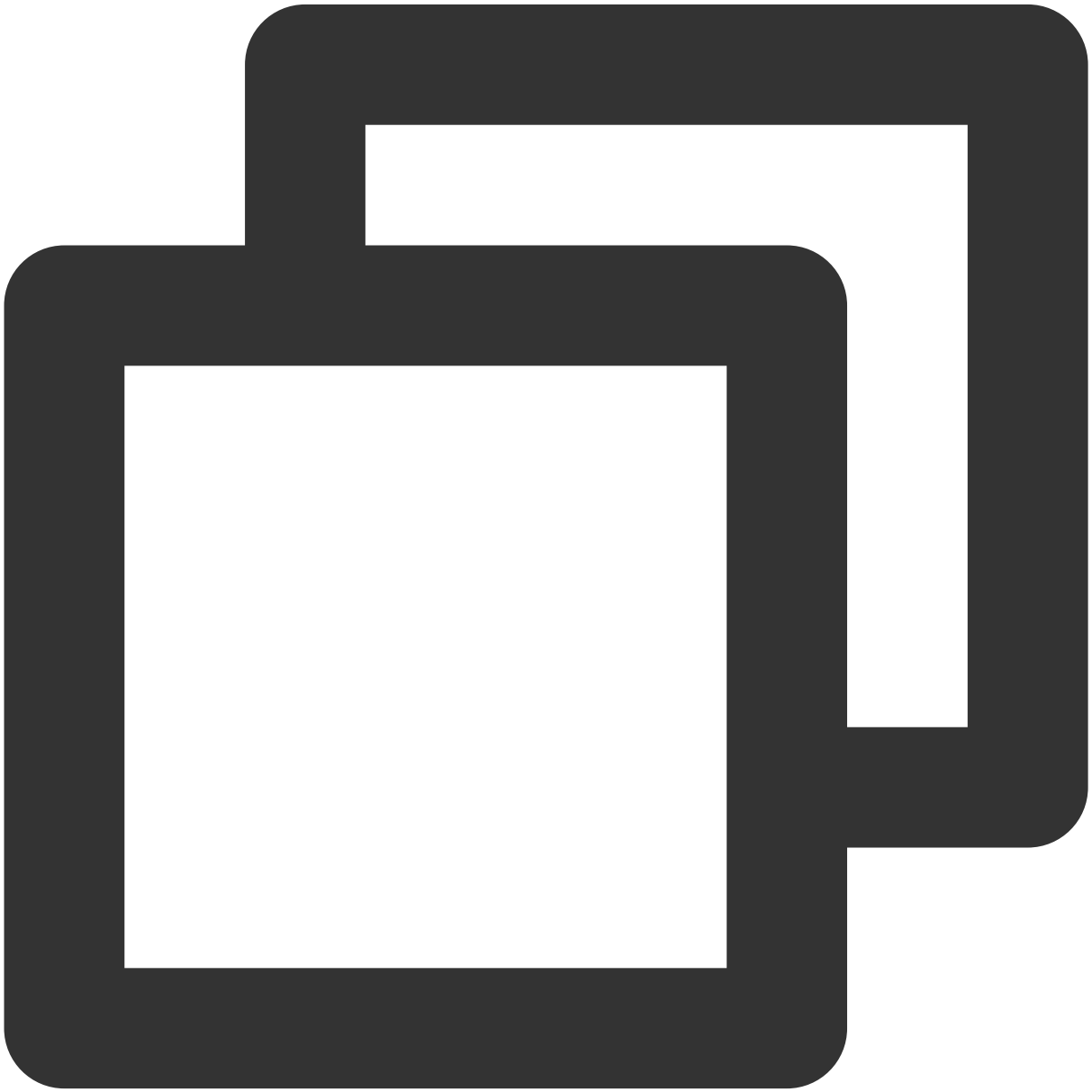


```
tim.deleteGroupMember(options);
```

라이브 룸에서 민감한 콘텐츠 필터링

라이브 룸에서 민감한 콘텐츠를 필터링하는 것은 다음과 같이 구현할 수 있는 또 다른 중요한 기능입니다.
1. 그룹 메시지를 보내기 전에 콜백을 바인딩합니다.

2. 콜백 데이터를 기반으로 메시지 유형을 식별하고 메시지 데이터를 Tianyu 또는 다른 타사 점검 서비스에 전달합니다.
3. 메시지가 일반 문자 메시지인 경우 Tianyu의 점검 결과를 기다렸다가 메시지를 IM 백엔드로 전달할지 여부를 나타내는 데이터 패킷을 반환합니다.
4. 메시지가 리치 미디어 메시지인 경우 메시지를 IM 백엔드로 전달하기 위한 데이터 패킷을 반환합니다. Tianyu가 비동기 결과를 반환한 후 메시지가 규정 위반으로 식별되면 메시지 편집 API 또는 사용자 정의 그룹 메시지 API를 통해 메시지 변경 알림을 전달합니다. 알림을 받은 후 클라이언트는 규정 위반 메시지를 차단합니다.
메시지 전송 전 콜백의 예시 데이터:

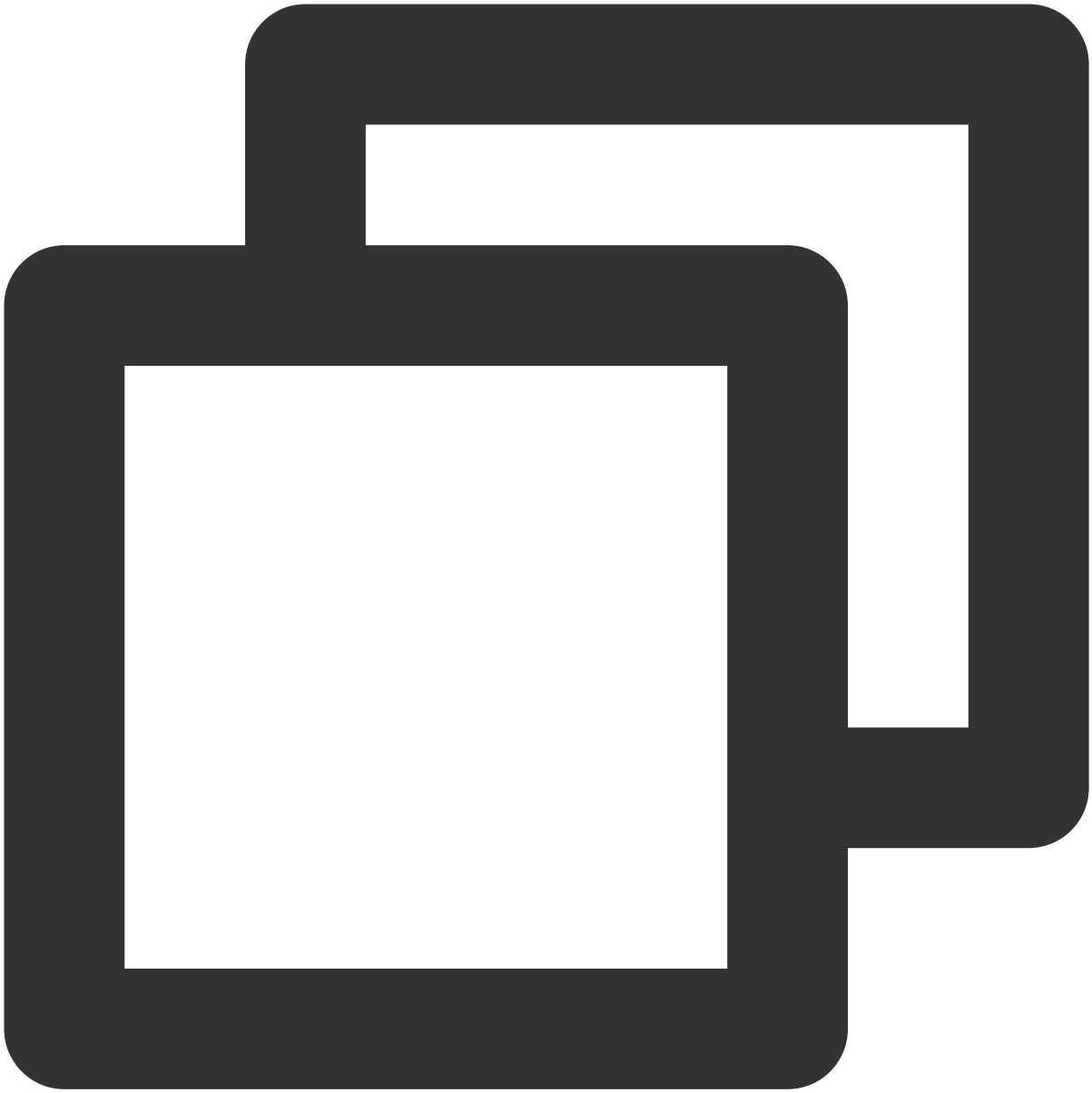


```
{
```

```
    "CallbackCommand": "Group.CallbackBeforeSendMsg", // 콜백 명령
    "GroupId": "@TGS#2J4SZEAEL", // 그룹 ID
    "Type": "Public", // 그룹 유형
    "From_Account": "jared", // 발신자
    "Operator_Account":"admin", // 요청 개시자
    "Random": 123456, // 랜덤 숫자
    "OnlineOnlyFlag": 1, // 값은 온라인 메시지인 경우 1이고 그렇지 않은 경우 0(기본값)입니다
    "MsgBody": [ // 메시지 본문, 자세한 내용은 TIMMessage 메시지 객체 참고
        {
            "MsgType": "TIMTextElem", // 텍스트
            "MsgContent": {
                "Text": "red packet"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data"
}
```

MsgBody의 MsgType 필드를 기반으로 메시지 유형을 식별할 수 있습니다. 필드에 대한 자세한 내용은 Callback Before Sending a Group Message를 참고하십시오.

규정 위반 메시지를 특정 방식으로 처리하도록 선택할 수 있으며, 이는 IM 백엔드로 반환되는 콜백의 데이터 패킷을 통해 구현할 수 있습니다.

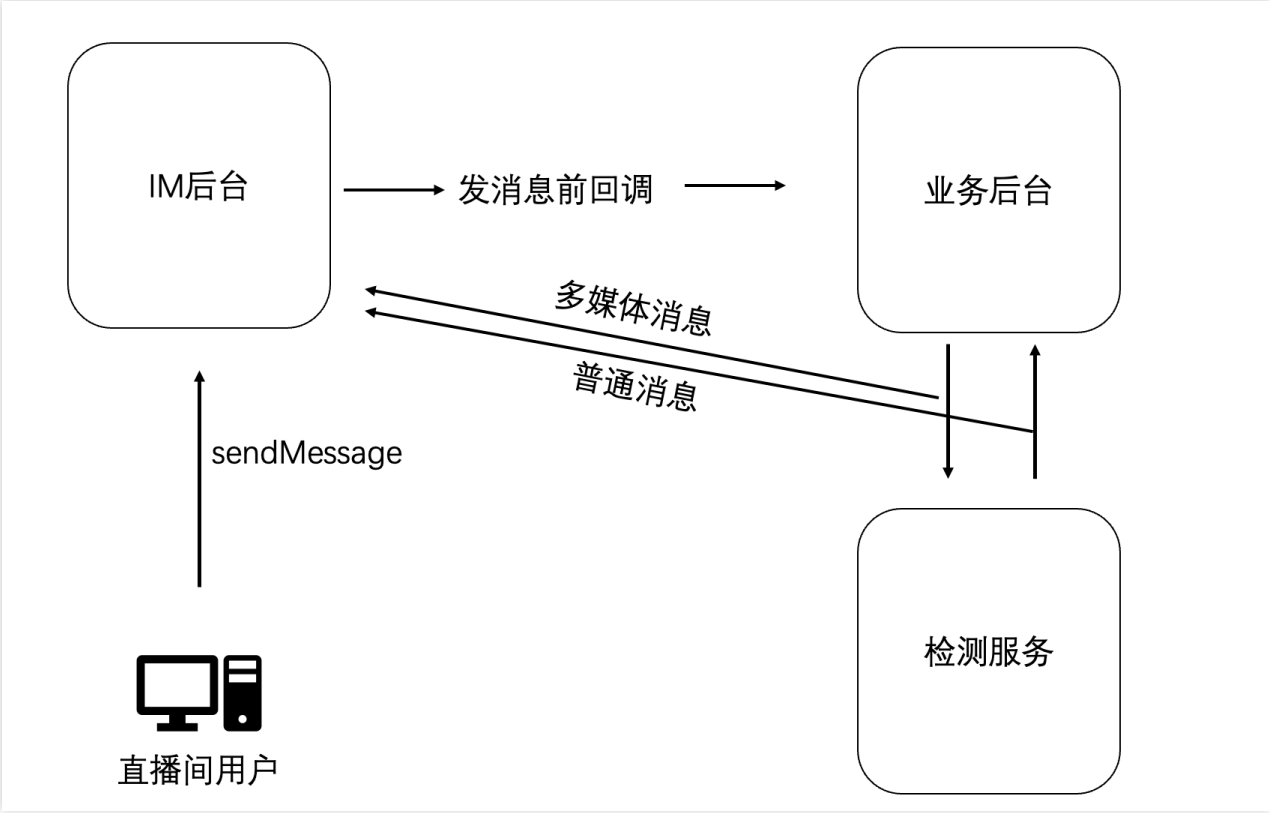


```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 다른 ErrorCode 값은 다른 의미를 갖습니다.
}
```

ErrorCode	설명
0	발언 허용, 메시지 전달 가능
1	발언 거부, 클라이언트 10016 반환

2	자동 폐기가 활성화되고 클라이언트가 메시지를 정상적으로 반환

필요에 따라 사용할 수 있습니다.
민감한 콘텐츠 감지 순서도는 다음과 같습니다.



라이브 룸의 그룹 구성원 목록

`getGroupMemberList` API를 호출하여 표시할 라이브 룸의 온라인 그룹 구성원 목록을 가져올 수 있습니다. 오디오/비디오 그룹(AVChatRoom)은 구성원이 많기 때문에 전체 구성원 목록 풀링 기능은 사용할 수 없습니다. 플래그십 에디션과 그 외의 에디션은 설정이 다릅니다.
1. 플래그십 에디션이 아닌 사용자는 `getGroupMemberList` 를 호출하여 최근 30명의 그룹 구성원을 풀링할 수 있습니다.
2. 플래그십 에디션 사용자는 `getGroupMemberList` 를 호출하여 최근 1,000명의 그룹 구성원을 풀링할 수 있습니다. 이 기능은 IM 콘솔에서 활성화해야 합니다. 활성화되지 않은 경우 플래그십 이외 기타 에디션에서와 같이 최근 30명의 그룹 구성원만 풀링합니다.
콘솔의 구성은 다음 이미지 2.6과 같습니다.

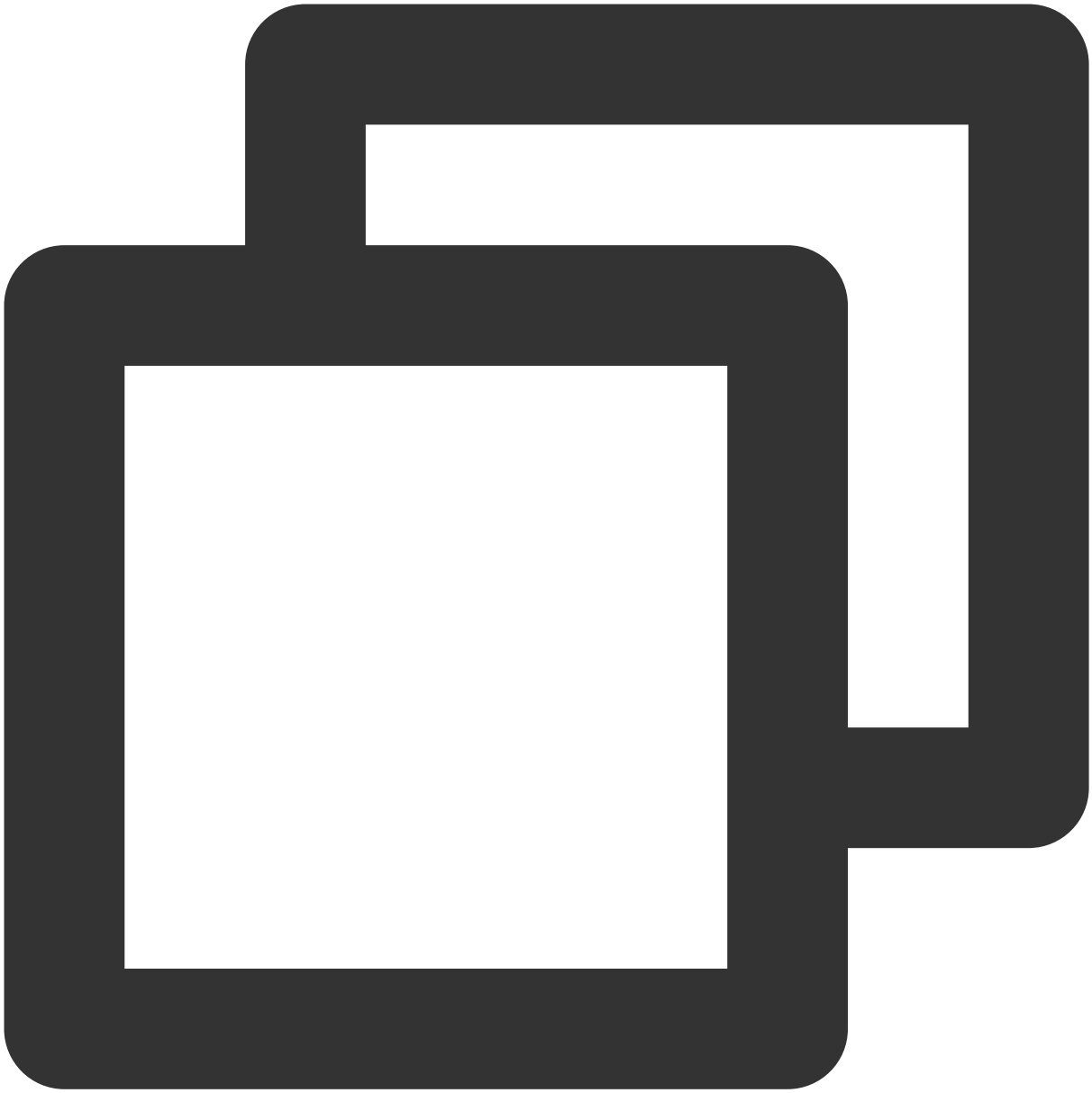
플래그십 에디션 사용자가 아닌 경우 `getGroupMemberList` 와 그룹 수신의 onGroupMemberEnter 및 onGroupMemberQuit 콜백을 통해 클라이언트의 온라인 그룹 구성원 목록을 유지할 수 있습니다. 단, 사용자가 라이브룸을 나간 후 다시 입장한 후에는 최근 30명의 그룹 구성원만 풀링할 수 있습니다.

Android

iOS&Mac

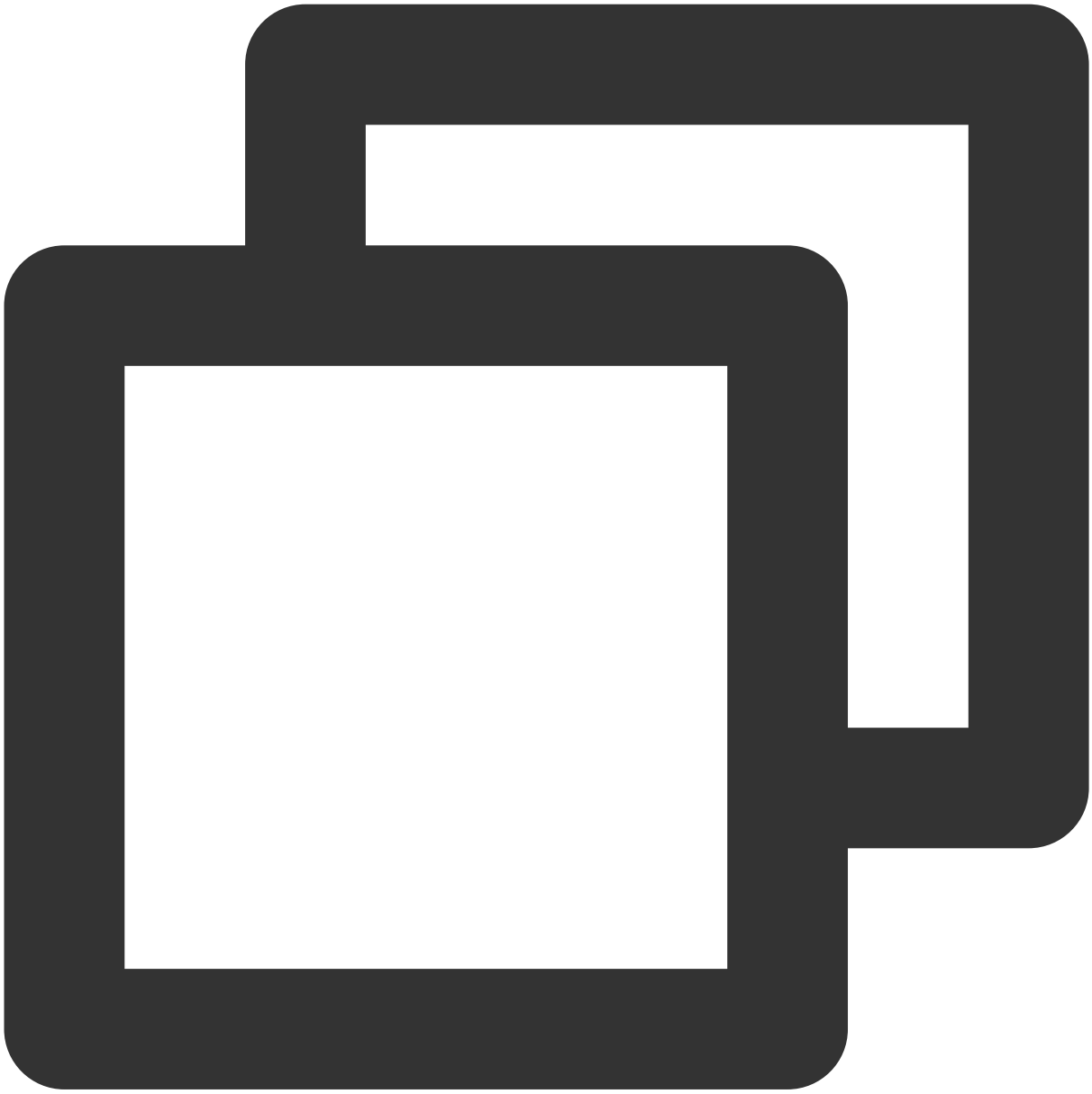
Flutter

Web

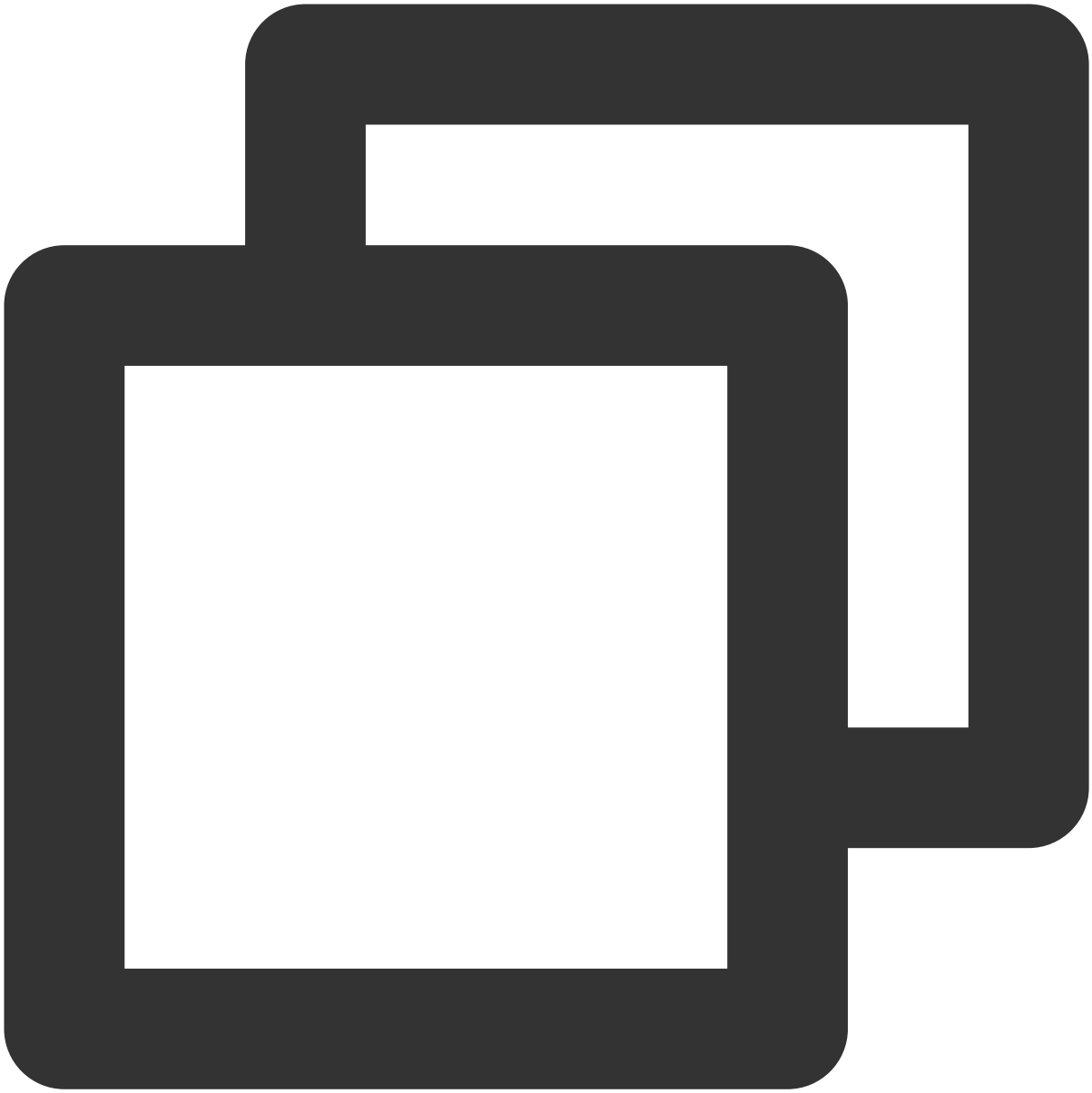


```
// filter 매개변수를 통해 그룹 소유자의 프로필을 풀링하도록 지정
int role = V2TIMGroupMemberFullInfo.V2TIM_GROUP_MEMBER_FILTER_OWNER;
V2TIMManager.getGroupManager().getGroupMemberList("testGroup", role, 0,
    new V2TIMValueCallback<V2TIMGroupMemberInfoResult>() {
    @Override
    public void onError(int code, String desc) {
        // 메시지 가져오기 실패
    }

    @Override
    public void onSuccess(V2TIMGroupMemberInfoResult v2TIMGroupMemberInfoResult) {
```

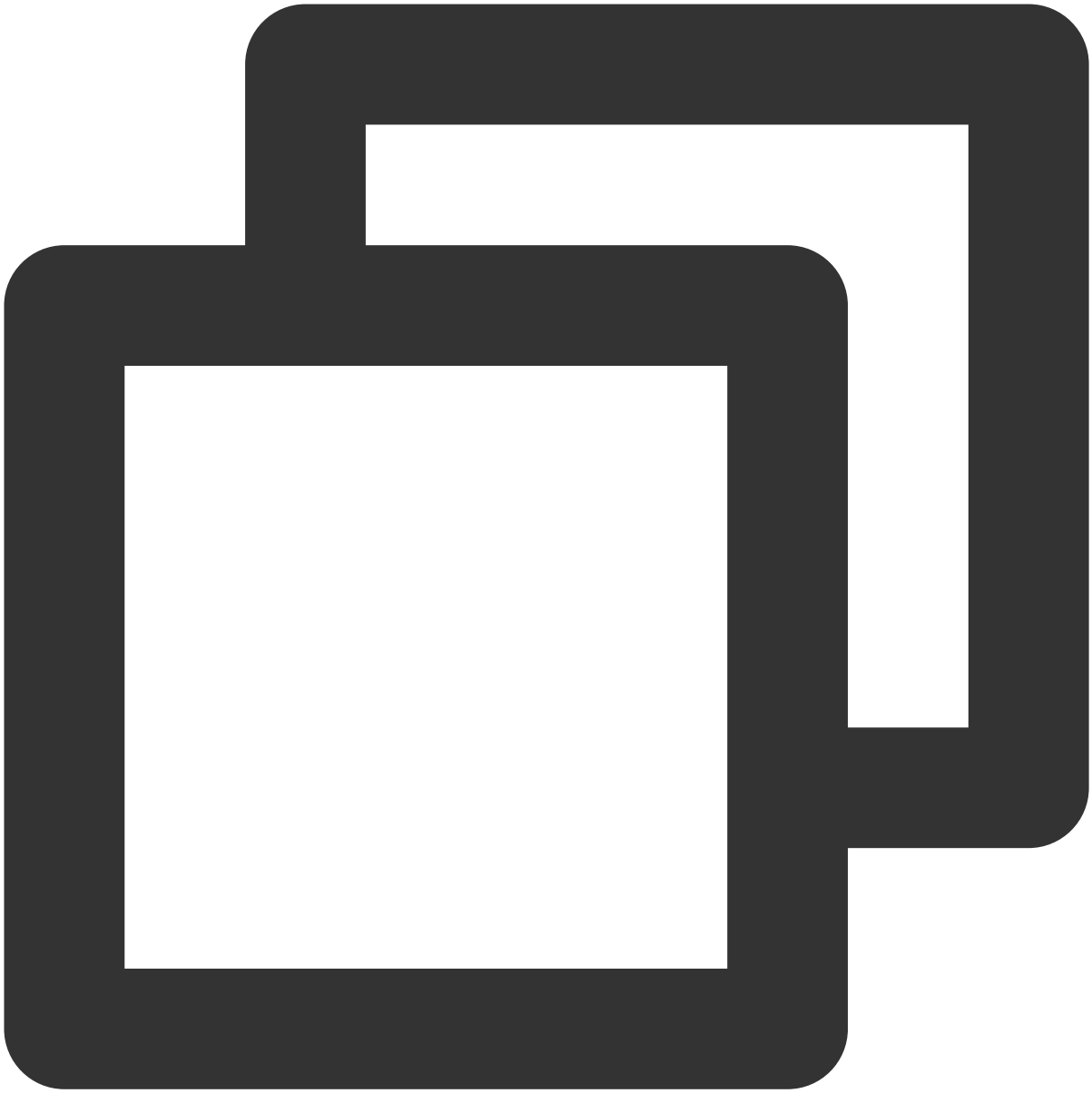
```
        // 대화 가져오기 성공
    }
});
```



```
[[V2TIMManager sharedInstance] getGroupMemberList:@"groupA" filter:V2TIM_GROUP_MEMB
    // 대화 가져오기 성공
} fail:^(int code, NSString *desc) {
    // 메시지 가져오기 실패
}];
```



```
// filter 매개변수를 사용하여 그룹 소유자의 프로필만 가져오도록 지정합니다. filter를 All로 설
groupManager.getGroupMemberList(count: 10,filter: GroupMemberFilterTypeEnum.V2TIM_G
```



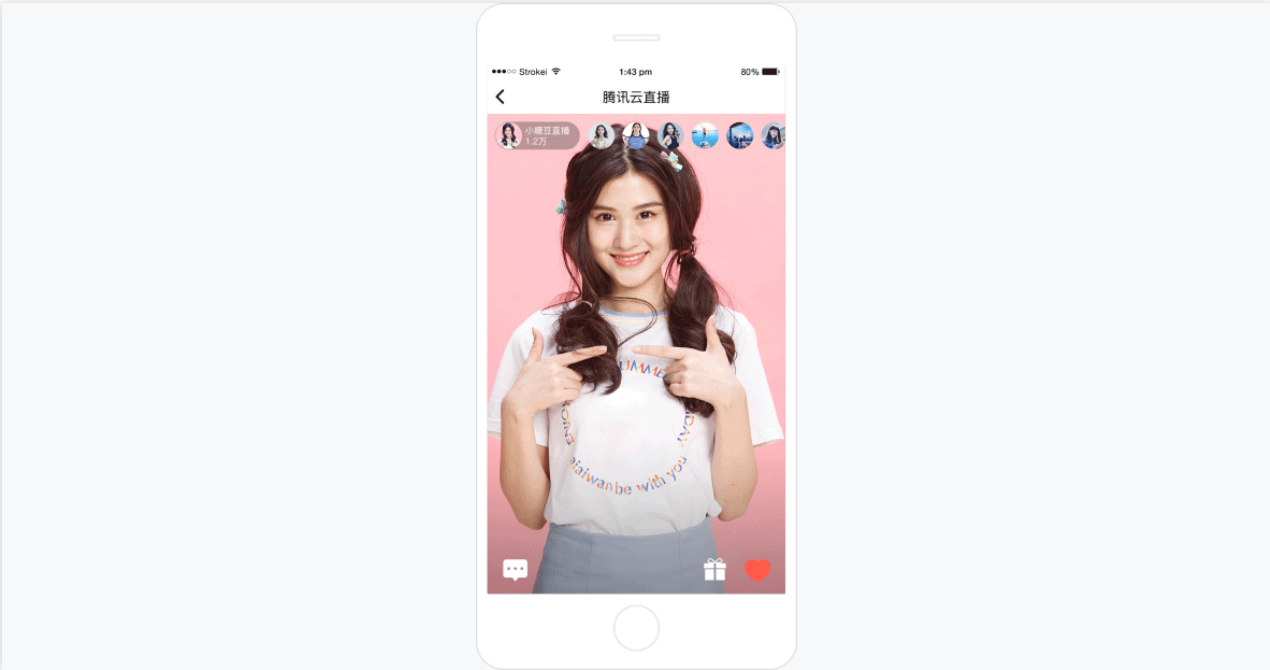
```
tim.getGroupMemberList(options);
```

라이브 룸의 화면 댓글을 기반으로 한 경품 추첨

메시지 통계와 마찬가지로 라이브 스트리밍의 경품 추첨도 메시지를 보낸 후 콜백이 필요합니다. 구체적으로, 메시지 내용이 감지되고 경품 추첨의 키워드에 도달한 사용자가 풀에 추가됩니다.

실시간 화면 댓글

AVChatRoom(오디오/비디오 그룹)은 친근한 인터랙션 경험을 구축하기 위해 화면 댓글, 선물하기 및 좋아요 등 다양한 메시지를 지원합니다.



라이브 룸에서 방송

방송 기능은 라이브 룸의 시스템 알림 기능과 유사하지만 메시징에 속한다는 점에서 후자와 다릅니다. 시스템 관리자가 방송 메시지를 전달하면 SDKAppID 아래의 모든 라이브 룸이 이를 수신합니다.
방송 기능은 현재 플래그십 에디션에서만 사용할 수 있으며 콘솔에서 활성화해야 합니다.
비즈니스 백엔드에서 방송 메시지를 보내는 방법에 대한 자세한 지침은 오디오/비디오 그룹 방송 메시지를 참고하십시오.
설명：
플래그십 에디션 사용자가 아닌 경우 서버에서 사용자 정의 그룹 메시지를 보내 기능을 구현할 수 있습니다.

3. 라이브 오디오/비디오 스트림

앵커(스트림 푸시)

현재 CSS는 스트림을 푸시하는 다음과 같은 방법을 제공합니다.
1.1 클라이언트에서 OBS 푸시 스트림
1.2 Web에서 Web 푸시
1.3 App에서 MLVB SDK
설명：

앵커가 스트림을 푸시할 푸시 주소를 구성해야 하며, 이는 푸시 스트림 설정에 따라 생성하거나 주소 생성기를 통해 생성할 수 있습니다.

클라이언트(풀 스트림)

클라이언트는 다양한 방법으로 라이브 스트림을 얻을 수 있습니다.

1.1 PC에서 VCL 푸시 스트림

1.2 App용 MLVB SDK

설명：

스트림을 가져오려면 재생 주소를 구성해야 하며, 이는 재생 설정을 따라 생성하거나 주소 생성기를 통해 생성할 수 있습니다. SDK 액세스 과정에 대한 자세한 내용은 재생을 참고하십시오.

라이브 디렉터

라이브 스트리밍에 대한 더 많은 요구 사항을 충족하기 위해 라이브 디렉터를 사용하여 라이브 스트림을 처리할 수 있습니다. 현재 라이브 디렉터는 다음 기능을 지원합니다.

1.1 워터마크, 텍스트 및 전환

1.2 라이브 스트리밍을 위한 VOD

1.3 라이브 스트리밍을 위한 정의, 비트레이트 및 해상도 설정

1.4 라이브 스트리밍을 위한 레이아웃 조정

1.5 라이브 녹화

1.6 실시간 화면 댓글 추가

1.7 라이브 스트림 모니터링

고급 기능 더보기

CSS는 푸시/풀 스트림 서비스 외에도 다음과 같은 고급 기능이 있습니다.

1.1 라이브 리먹싱 및 트랜스코딩, 독점적인 초고속 고화질 트랜스코딩 기술로, 시나리오 동적 인코딩을 지능적으로 인식하고 라이브 고화질 저코드 및 화질 향상을 실현합니다.

1.2 타임 시프트, 라이브 방송 중 하이라이트 영상을 일시 중지 및 다시 볼 수 있도록 지원합니다.

1.3 라이브 방송 녹화, 라이브 방송 중 동기화 녹화 및 저장이 가능하여 녹화된 비디오의 후속 편집 및 재전파에 편리합니다.

1.4 라이브 방송 워터마크, 생방송 화면에 선명한 워터마크 또는 디지털 워터마크를 겹쳐 영상 도난 방지 효과를 실현합니다.

1.5 클라우드 혼합 스트림, 설정한 혼합 스트림 레이아웃에 따라 다양한 경로의 입력 소스를 새로운 스트림으로 동기식으로 혼합하여 라이브 인터랙션 효과를 실현할 수 있습니다.

1.6 릴레이, 다른 플랫폼의 라이브 또는 VOD 비디오를 라이브 방송 형태로 배포할 수 있으며, 컨텐츠 풀링 및 푸시 기능을 제공하고,기존 라이브 방송이나 영상을 대상 주소로 푸시합니다.

4. FAQ

1. CSS를 테스트할 수 있나요?

CSS 서비스를 활성화한 신규 사용자에게는 1년간 유효한 20GB의 트래픽이 무료로 제공됩니다. 이때 테스트에 의해 발생하는 트래픽은 이를 이용하여 차감할 수 있으며, 패키지를 초과하는 부분은 후불 일 결산 방식으로 과금됩니다. 동시에 라이브 방송 워터마크, 트랜스코딩, 녹화, 화면 캡처, 음란물 감지 등과 같은 라이브 방송 부가 기능을 사용할 수도 있습니다. 부가 기능은 기본적으로 비활성화되어 있으며, 필요에 따라 사용 및 과금됩니다. 기능에 대한 자세한 내용은 제품 개요를 참고하십시오.

2. 라이브 방송은 접속 인원 수 제한이 있습니까?

기본적으로 CSS는 네트워크 및 기타 조건이 허용하는 한 라이브 스트림의 온라인 시청자 수를 제한하지 않습니다. 그러나 대역폭 제한을 설정한 경우 기존 시청자가 너무 많아 대역폭 제한을 초과하면 새로운 시청자가 라이브 스트림을 시청할 수 없습니다. 이 경우 온라인 시청자 수에 제한이 있습니다.

3. 메시지 전송 중 메시지 전송 상태, `Message.nick` , `Message.avatar` 필드가 비어 있는 경우 UI에 닉네임과 프로필 사진을 표시하려면 어떻게 해야 하나요?

getUserInfo API를 호출하여 사용자 정보에서 nick 및 avatar 필드를 가져와 메시지 전송 필드로 사용합니다.

4. 메시지가 손실되는 원인은 무엇입니까?

메시지가 손실되는 가능 원인은 다음과 같습니다.
오디오-비디오 그룹에는 초당 40개 메시지의 빈도 제한이 적용됩니다. 메시지 전송 전 콜백과 메시지 전송 후 콜백 수신 여부를 확인할 수 있습니다. 전자는 수신되었지만 후자는 수신되지 않은 경우 빈도 제한으로 인해 메시지가 차단된 것입니다.
자세한 내용은 메시지를 참고하십시오. Web 클라이언트 퇴장으로 인해 Android/iOS/PC 클라이언트가 퇴장되는지 확인합니다.
Web 클라이언트에서 문제가 발생하면 SDK 버전이 v2.7.6 이전인지 확인하십시오. 그렇다면 최신 버전으로 업그레이드하십시오.
상기 원인에 해당되지 않는 경우, 티켓 제출을 통해 문의주시기 바랍니다.

5. 고객센터

그룹 번호 검색: 853084820, 더 자세한 답변을 드리겠습니다.

Live Chat Room

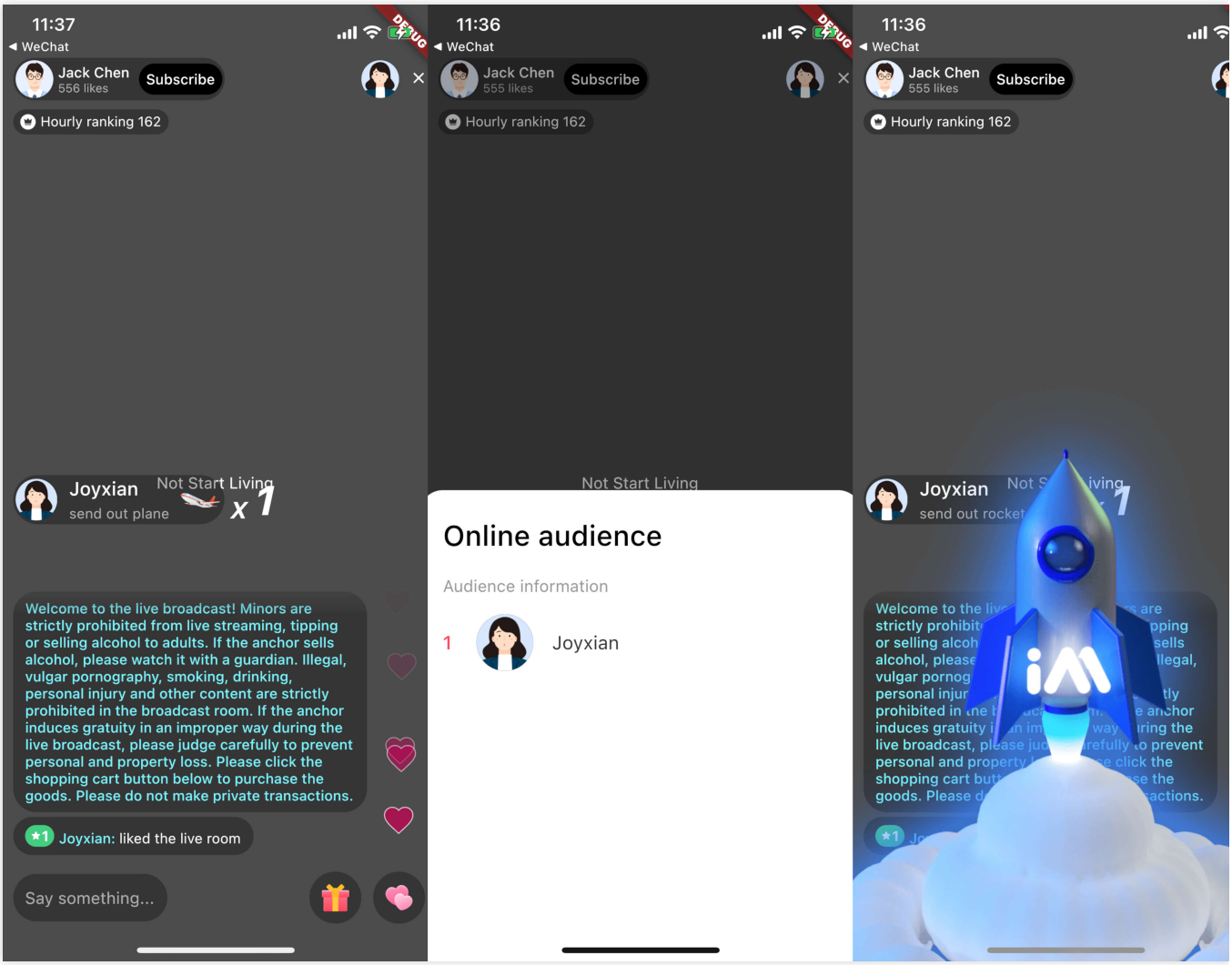
최종 업데이트 날짜: : 2024-05-13 17:58:07

Foreword

This article is based on the Tencent Cloud AV Chat Room plug-in, combined with the Live Flutter SDK Realized live broadcast business scenario.

Tencent Cloud AV Chat Room

`Tencent Cloud AV Chat Room` is a UI component based on Tencent Cloud Chat SDK and implemented around chat interaction scenarios.
You can use this component to quickly implement an application with **tens of millions of interactions**.



Introduction

Preconditions

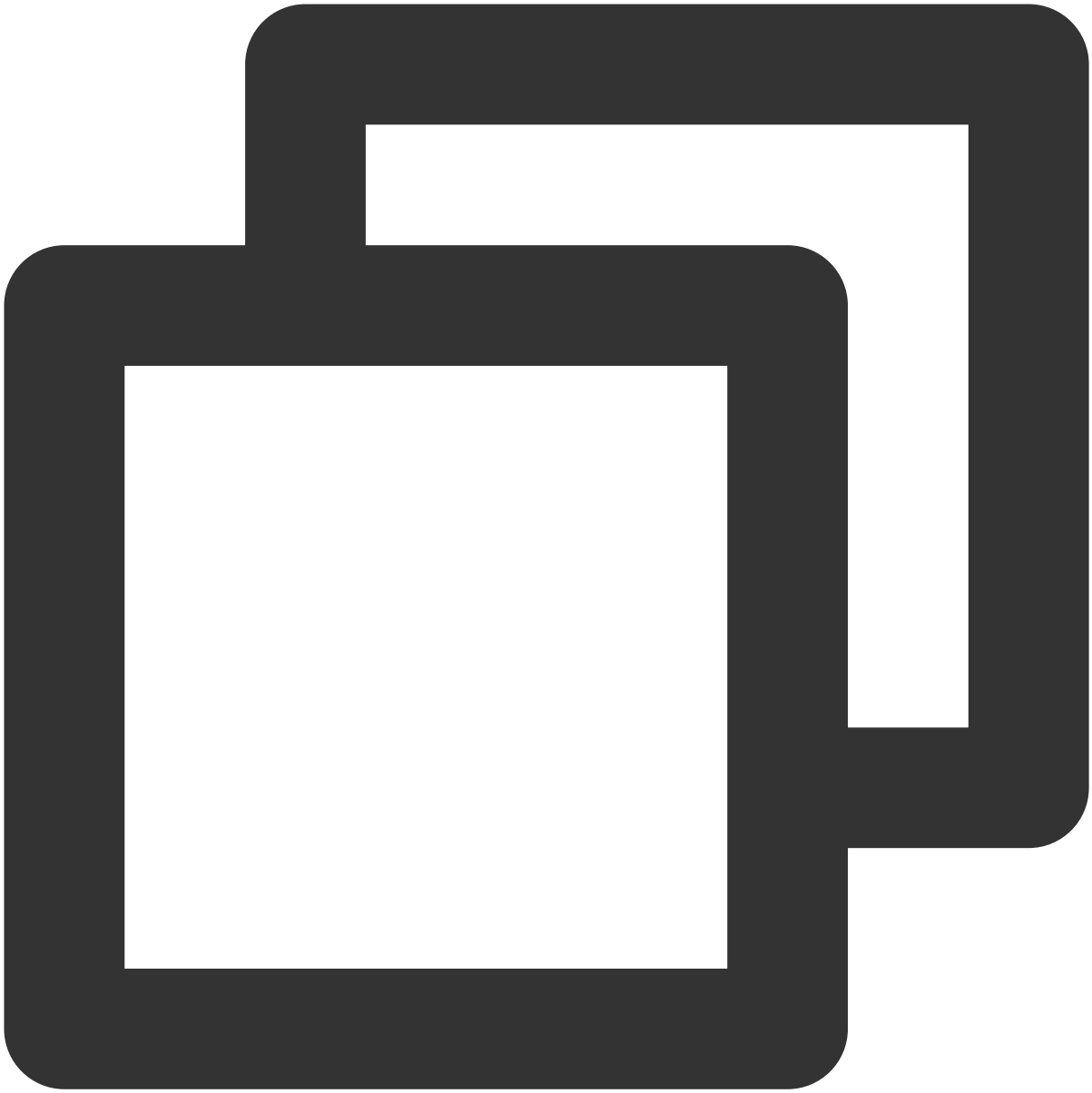
Have registered Tencent Cloud account and completed [identity verification] (https://www.tencentcloud.com/zh/document/ product/378/3629).

Create an application by referring to Create and upgrade an application, and record `SDKAppID` .

Select your application in IM Console, click Accessibility Tools -> UserSig Generation & Verification in the left navigation bar, and create a UserID and its corresponding `UserSig` , copy the three `UserID` , `Signature (Key)` , `UserSig` , which will be used in subsequent logins.

Create a Flutter app.

Add tencent_cloud_av_chat_room in the pubspec.yaml file under dependencies. Or execute the following command:



```
/// step 1:
flutter pub add tencent_cloud_av_chat_room

/// step 2:
flutter pub get
```

Step 1: Initialize and login to IM

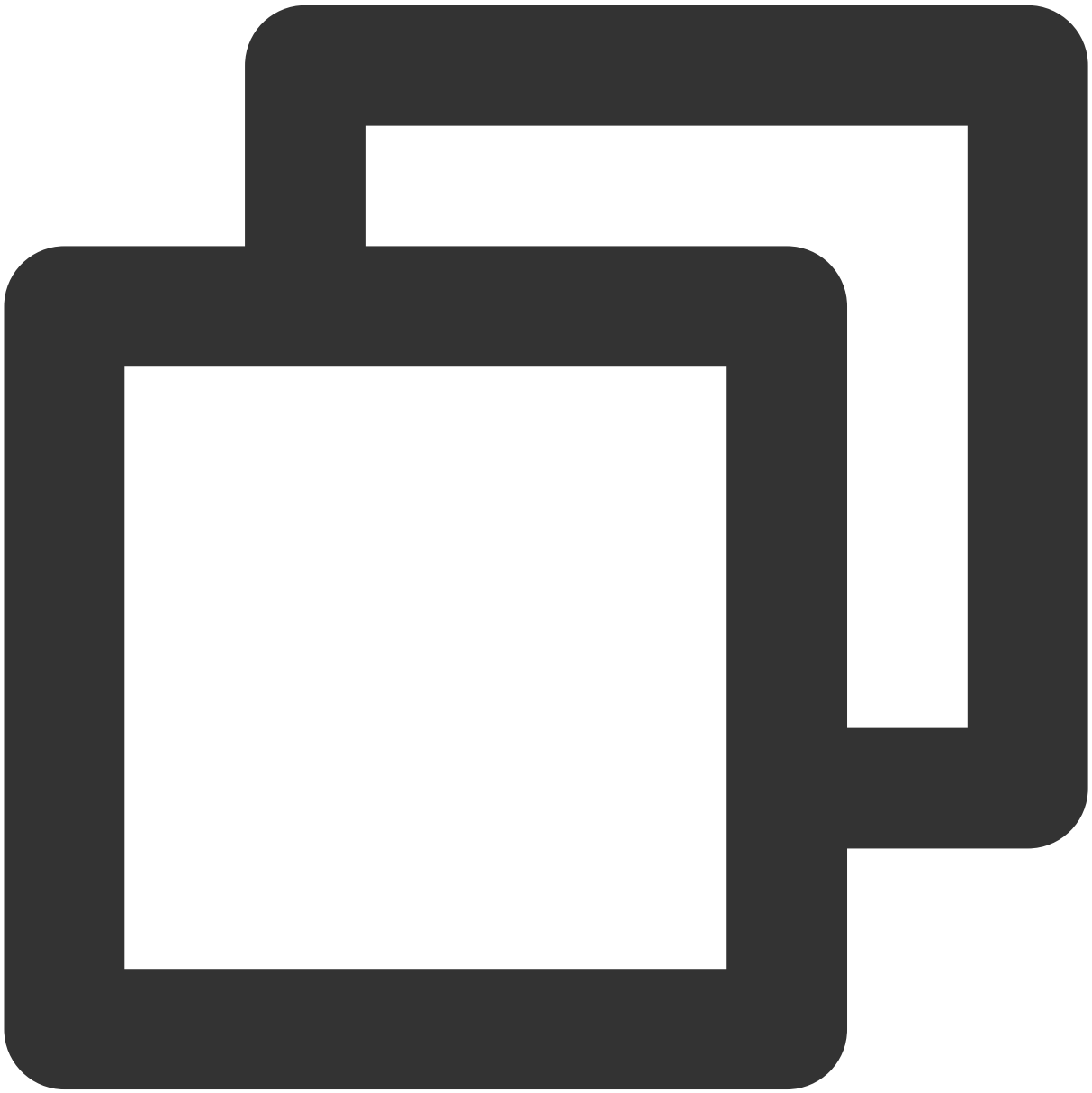
There are two ways to initialize and log in to IM:

External component: The entire application is initialized and logged in only once.

Inside the component: pass parameters into the component through configuration.
If you are integrating this plugin in an existing IM flutter project, you can skip this step.

Outside the component (recommended)

Initialize IM in the flutter app you created, note that the IM app only needs to be initialized once. This step can be skipped if integrating in an existing IM project.

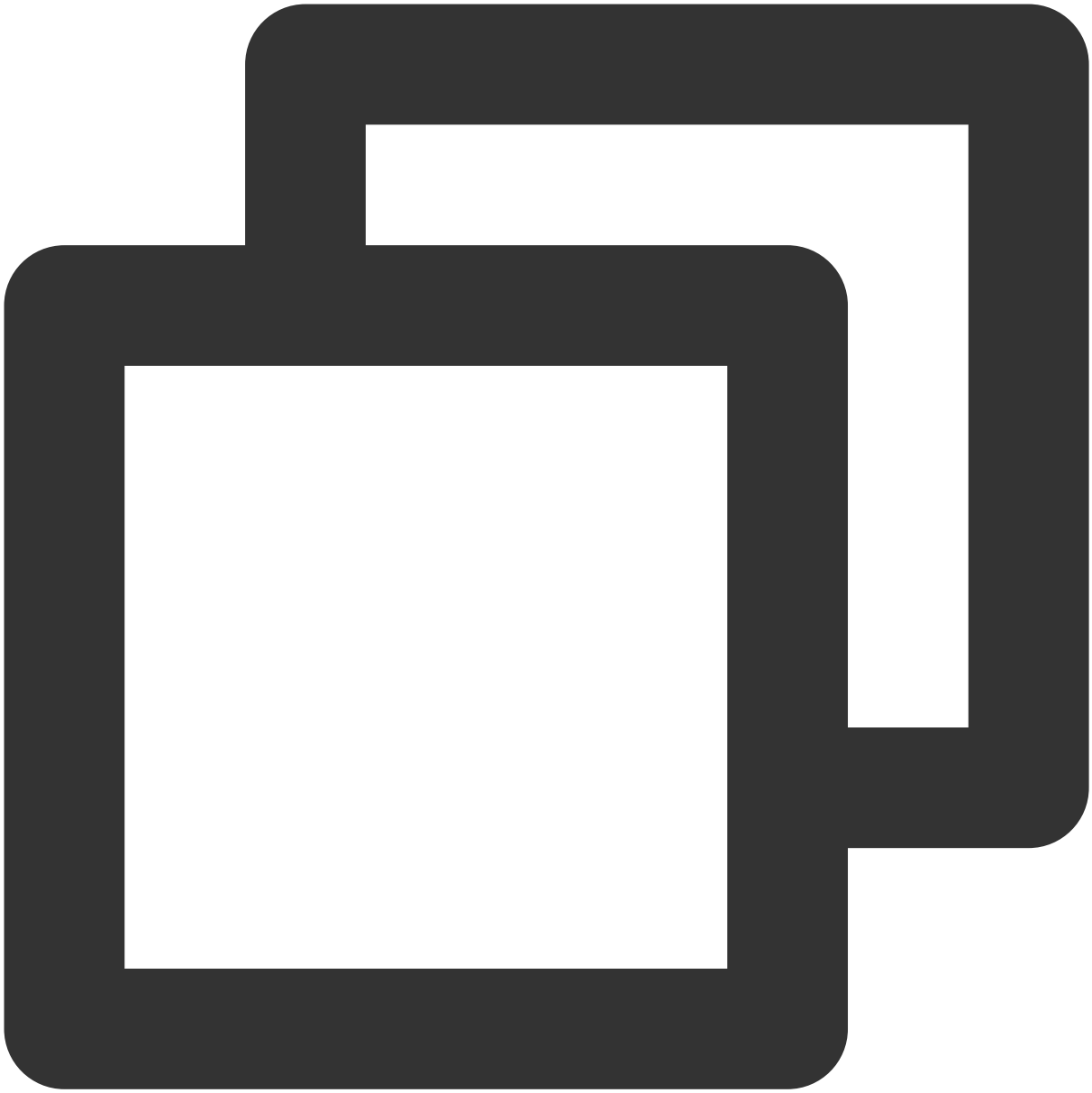


```
import 'package:tencent_av_chat_room_kit/tencent_cloud_chat_sdk_type.dart';

class _MyHomePageState extends State<MyHomePage> {
```

```
    final int _sdkAppID = 0; // SDKAppID of the IM application created in the prec
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
  @override
  void initState() {
      super.initState();
      _initAndLoginIm();
    }

  _initAndLoginIm() async {
      await TencentImSDKPlugin.v2TIMManager.initSDK(
          sdkAppID: _sdkAppID,
          loglevel: LogLevelEnum.V2TIM_LOG_ALL,
          listener: V2TimSDKListener());
      TencentImSDKPlugin.v2TIMManager
        .login(userID: _loginUserID, userSig: _userSig);
  }
}
```

Inside the component

You can also pass `SDKAppID`, `UserSig`, `UserID` into the component through configuration to initialize and log in IM.

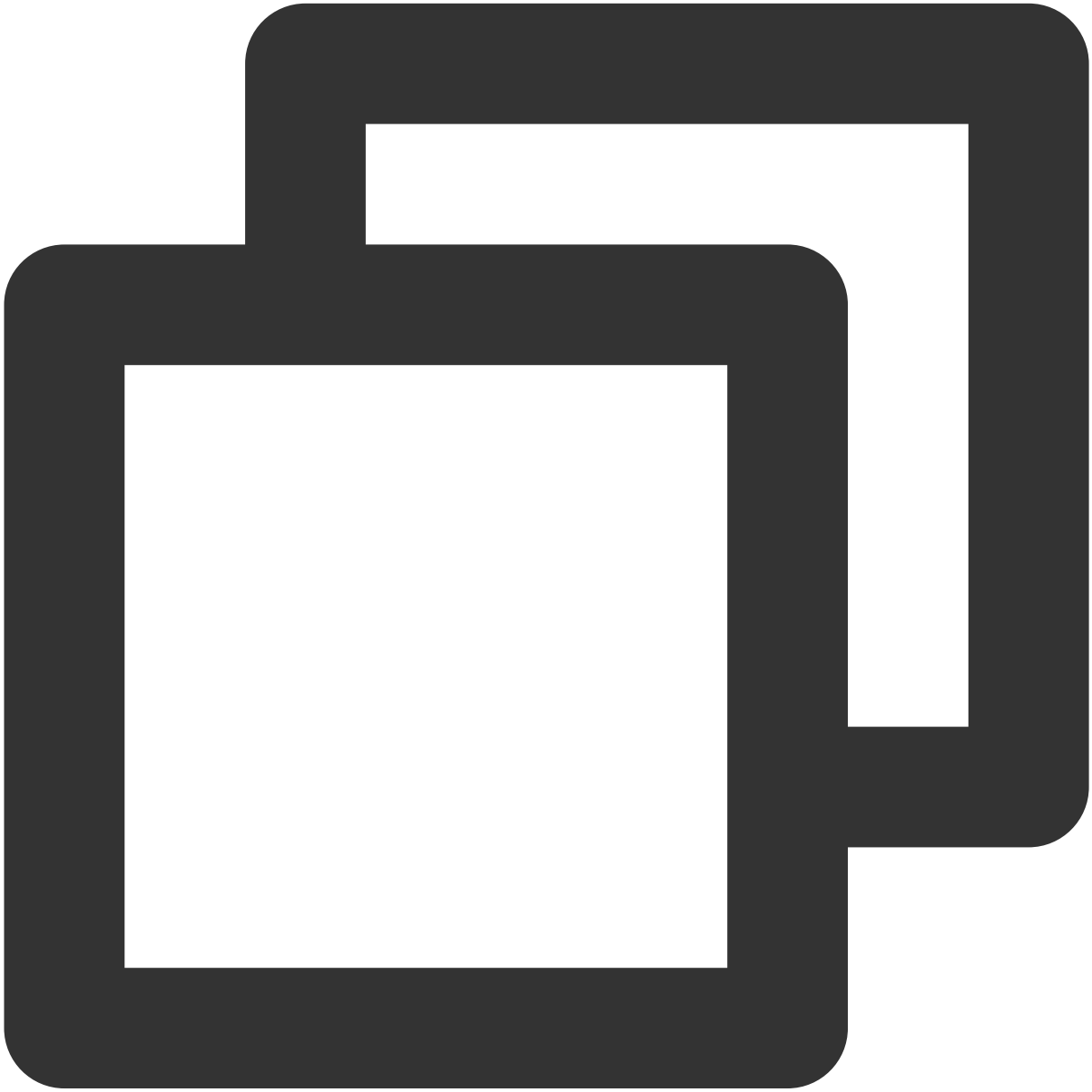


```
import 'package:tencent_cloud_av_chat_room/tencent_cloud_av_chat_room.dart';

class _TencentAVChatRoomKitState extends State<TencentCloudAVChatRoom> {
    final int _sdkAppID = 0; // SDKAppID of the IM application created in the preco
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
    @override
      Widget build(BuildContext context) {
        return TencentCloudAVChatRoom(
            config: TencentCloudAvChatRoomConfig(
                loginUserID: _loginUserID, sdkAppID: _sdkAppID, userSig: _userSig))
```

```
    }
}
```

Step 2: Using Components

Use that component in your appropriate module.



```
import 'package:tencent_cloud_av_chat_room/tencent_cloud_av_chat_room.dart';

class _TencentAVChatRoomKitState extends State<TencentCloudAVChatRoom> {
    final int _sdkAppID = 0; // SDKAppID of the IM application created in the preco
```

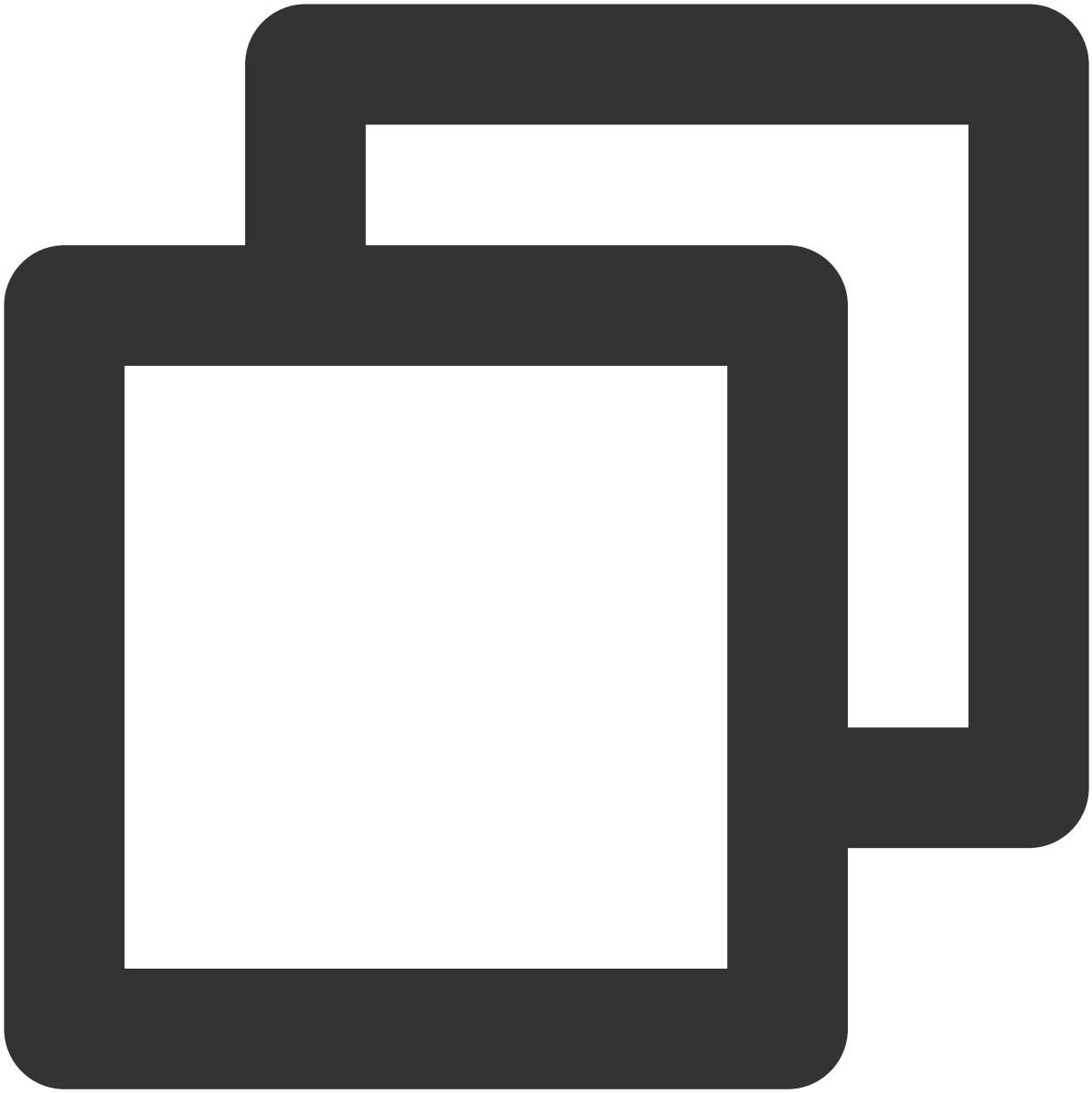
```
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
    @override
      Widget build(BuildContext context) {
        return TencentCloudAVChatRoom(
            data: TencentCloudAvChatRoomData(anchorInfo: AnchorInfo()),
            config: TencentCloudAvChatRoomConfig(
                loginUserID: _loginUserID, sdkAppID: _sdkAppID, userSig: _userSig,
    }
}
```

Cooperate with Tencent Cloud View Cube to build a live broadcast room

Live streaming is ubiquitous in life, and more and more companies and developers are building their own live streaming platforms. The following will introduce how to build a live broadcast room with Tencent Cloud View Cube. The live broadcast room mainly includes two parts: live broadcast and interaction. In the live broadcast part, we will use Tencent Cloud View Cube mobile live broadcast to realize live streaming, screen playback, etc. Interaction uses this plug-in to achieve live broadcast likes, gift giving, barrage sending and receiving, etc.
The relevant code of this article can be downloaded and viewed in Github.

Live streaming and screen playback

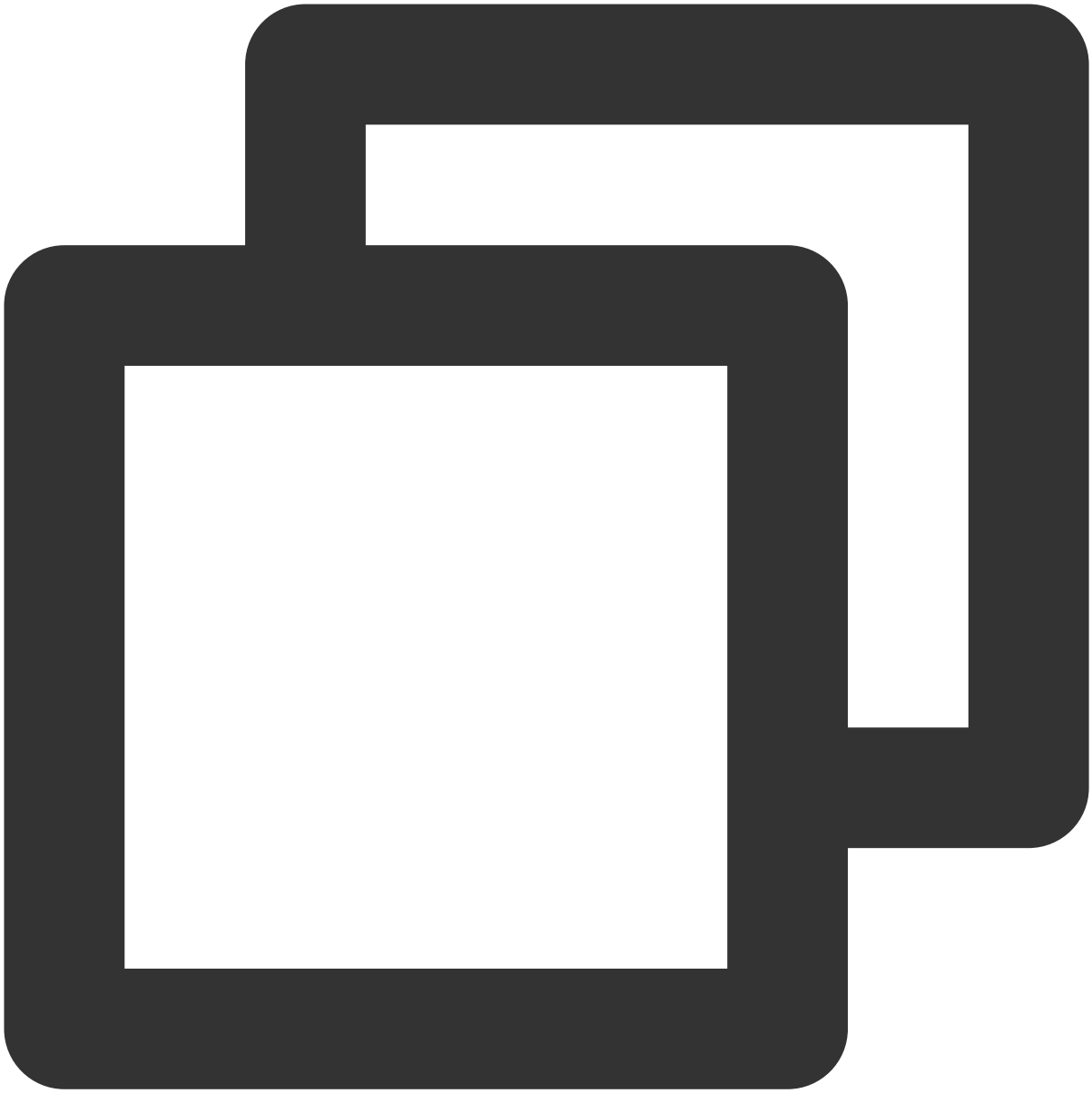
The live broadcast SDK is one of the sub-products of the Tencent Cloud View Cube product family. The live broadcast SDK supports live streaming push, pull streaming, host-audience interaction with hosts, host cross-room PK and other capabilities, providing users with stable and extremely fast live broadcast terminal services.
See Live Streaming Flutter SDK Standard Live Streaming to realize the functions of live streaming and screen playback.
1: Set License



```
// live.dart
class Live extends StatefulWidget {
  const Live({Key? key}) : super(key: key);

  @override
  State<Live> createState() => _LiveState();
}

class _LiveState extends State<Live> {
    ...
    @override
```

```
    void initState() {
        super.initState();
        setupLicense();
    }

    /// Tencent Cloud License Management Page (https://console.tencentcloud.com/liv
    setupLicense() {
        // The currently applied License LicenseUrl
        final licenseUrl = "";
        // License Key currently applied
        final licenseKey = "";
        V2TXLivePremier.setLicence(licenseUrl, licenseKey);
    }
    ...
}
```

2: Live streaming, screen playback



```
// live_player.dart
...
class _LivePlayState extends State<LivePlayer> {
    ...
    @override
    void initState() {
        super.initState();
        initPlayer();
    }
    initPlayer() async {
        _livePlayer = await V2TXLivePlayer.create();
```

```
    }

    @override
    void dispose() {
        super.dispose();
        _livePlayer.destroy();
    }

    @override
    Widget build(BuildContext context) {
        return Container(
            color: Colors.black.withOpacity(0.7),
            child: V2TXLiveVideoWidget(
                onViewCreated: (viewId) async {
                _localViewId = viewId;
                startPlay();
                },
            ),
        );
    }

    void startPlay() async {
        if (_isPlaying) {
            return;
        }
        if (_localViewId != null) {
            debugPrint("_localViewId $_localViewId");
            var code = await _livePlayer.setRenderViewID(_localViewId!);
            if (code != V2TXLIVE_OK) {
                debugPrint("StartPlay error： please check remoteView load");
            }
        }
        var url = widget.playUrl;
        debugPrint("play url: $url");
        var playStatus = await _livePlayer.startLivePlay(url);
        debugPrint("play status: $playStatus");
        if (playStatus != V2TXLIVE_OK) {
        setState(() {
            _onError = true;
        });
        debugPrint("play error: $playStatus url: $url");
            return;
        }
        await _livePlayer.setPlayoutVolume(100);
        setState(() {
            _isPlaying = true;
        });
```

```
        }
    }
```

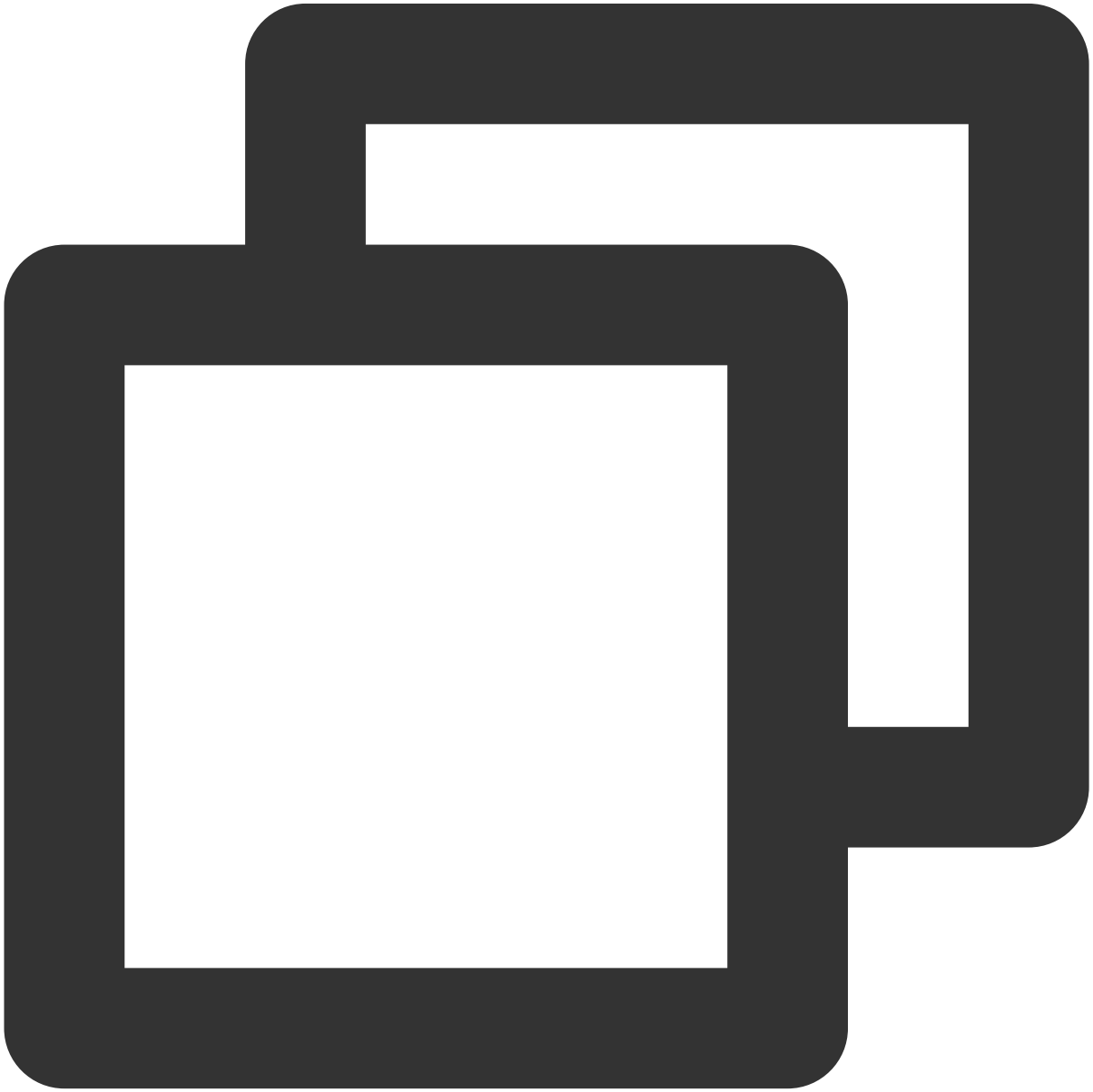
Live interaction (like, give gifts, send and receive barrage)

Live interaction is particularly important in live broadcast scenarios. Users interact with the anchor through barrage, gift giving, etc.

Next, see Introduction to integrate this plugin.

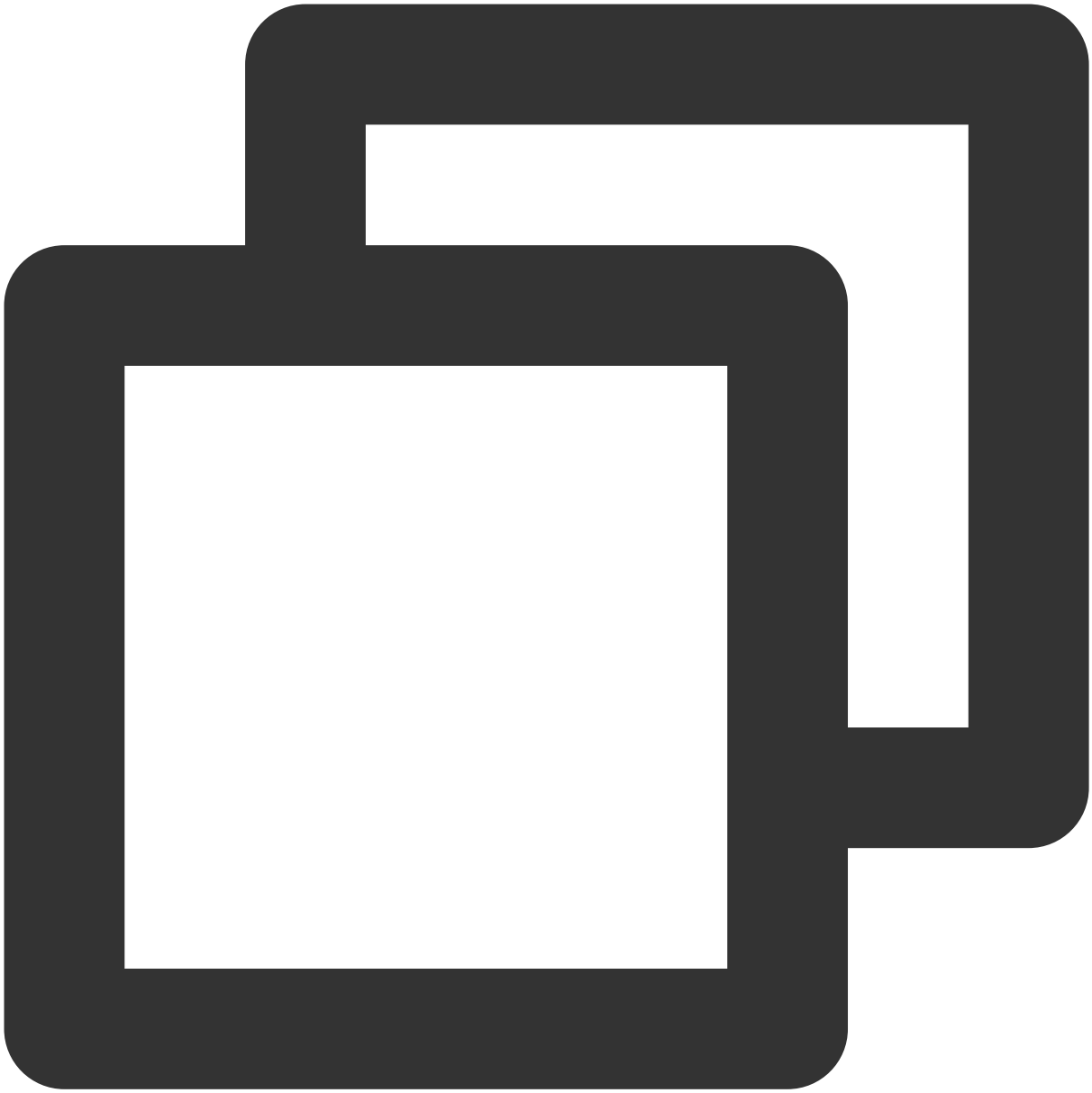
1: Initialize, log in to IM

We log in to IM outside the plugin, and usually the entire application only needs to initialize and log in to IM once.



```
import 'package:tencent_av_chat_room_kit/tencent_cloud_chat_sdk_type.dart';

class _MyHomePageState extends State<MyHomePage> {
     final int _sdkAppID = 0; // SDKAppID of the IM application created in the prec
    final String _loginUserID = ""; // UserID in precondition
    final String _userSig = ""; // UserSig in preconditions
    @override
    void initState() {
        super.initState();
        _initAndLoginIm();
      }

    _initAndLoginIm() async {
        await TencentImSDKPlugin.v2TIMManager.initSDK(
            sdkAppID: _sdkAppID,
            loglevel: LogLevelEnum.V2TIM_LOG_ALL,
            listener: V2TimSDKListener());
        TencentImSDKPlugin.v2TIMManager
          .login(userID: _loginUserID, userSig: _userSig);
    }
}
```

2: Use plugins



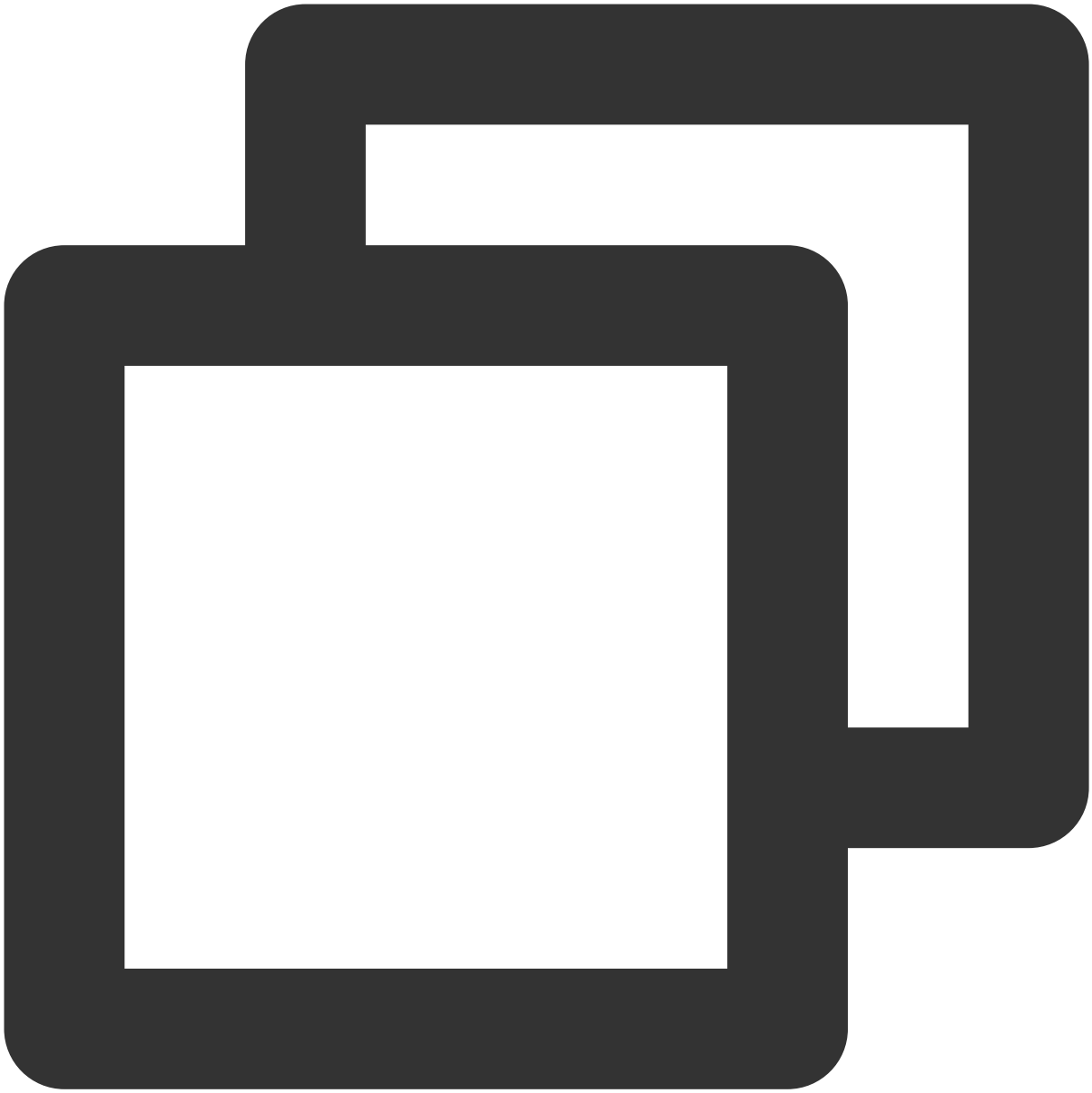
```
// live_room.dart

class LiveRoom extends StatelessWidget {
    final String loginUserID;
    final String playUrl;
    final String avChatRoomID = ''; // A live broadcast room is essentially all use

    LiveRoom({Key? key, required this.loginUserID, required this.playUrl})
      : super(key: key);

    @override
```

```
    Widget build(BuildContext context) {
        return Stack(
            children: [
                LivePlayer(playUrl: playUrl), // Live streaming and screen playback
                TencentCloudAVChatRoom(
                    config: TencentCloudAvChatRoomConfig(
                        avChatRoomID: avChatRoomID,
                        loginUserID: loginUserID, //Login user ID
                    ),
                    data: TencentCloudAvChatRoomData(
                        isSubscribe: false,
                        notification: "Broadcast Room Announcement",
                        anchorInfo: AnchorInfo(
                            subscribeNum: 200,
                            fansNum: 5768,
                            nickName: "stormy life",
                            avatarUrl:""
                        ),
                    )
                )
            ]
        )
    }
}
```

Above we use `Stack Widget` to combine `LivePlayer (live broadcast)` and `LiveRoom (interactive)` through cascading.

3: How to customize

The plug-in itself provides a default UI, but users often have different UIs during actual use. Next, we will introduce how to customize this plug-in.



```
...
@override
Widget build(BuildContext context) {
return Stack(
children: [
LivePlayer(playUrl: playUrl), // Live streaming and screen playback
TencentCloudAVChatRoom(
...
TencentCloudAvChatRoomCustomWidgets(
roomHeaderAction: Container(), // The area custom component is displayed in the upp
roomHeaderLeading: Container(), // The area custom component is displayed in the up
```

```
roomHeaderTag: Container(), // Below roomHeaderAction and roomHeaderLeading, it is
onlineMemberListPanelBuilder: (context, id) { // Customize the panel that expands a
return Container();
},
anchorInfoPanelBuilder: (context, id) { // Customize the expanded panel after the a
return Container();
},
giftsPanelBuilder: (context) { // Customize the panel displayed after clicking the
return Container();
},
messageItemBuilder: (context, message, child) { // Customize bullet chat messages
return Container();
},
messageItemPrefixBuilder: (context, message) { // Customize the prefix of bullet ch
return Container();
},
giftMessageBuilder: (context, message) { // Custom gift message, gift message that
return Container();
},
textFieldActionBuilder: (// Customize the lower right area of the screen
context,
) {
return [Container()];
},
textFieldDecoratorBuilder: (context) { // Customize the input box at the bottom lef
return Container();
}
)
)
]
)
}
```

4: Gift

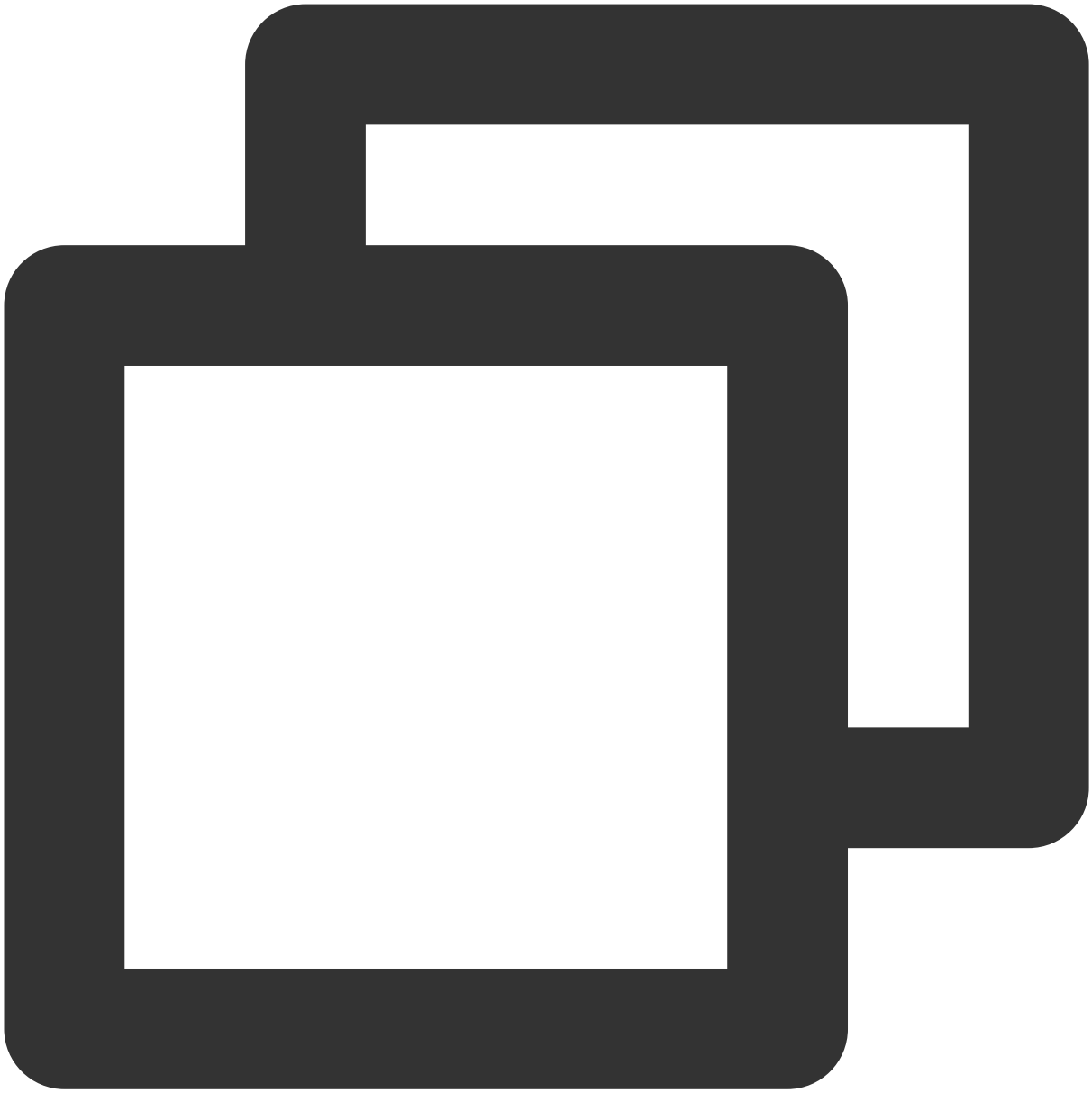
Giving gifts is a very important scene in the live broadcast scene. This plugin provides analysis of three types of gifts by default, users only need to send a custom message in a specific format to display the gift message on the interface. The default gift panel of this plug-in is only used to display the gift message parsing function. Usually, gifts will involve the logic of user billing and measurement, so users need to call the server interface to send gifts according to their business needs. information. The gift giving process is as follows:

The short connection request from the client to its own business server involves billing logic.

After billing, the sender directly sees that XXX sent XXX a gift. (To ensure that the sender sees the gift they sent, when there is a large amount of messages, the abandonment strategy may be triggered)

After billing and settlement, call the server interface to send a custom message (gift)

Gift Messages We do this by sending custom messages. Three gift formats are defined as follows:



```
// Gifts with special effects (the details of the gift will slide into the left sid
final customInfoRocket = {
"version": 1.0, // protocol version number
"businessID": "flutter_live_kit", // Business ID field
"data": {
"cmd":
"send_gift_message", // must be send_gift_message
"cmdInfo": {
"type": 3, // gift type
"giftUrl": "", // URL of gift image
"giftCount": 1, // number of gifts
```

```
"giftSEUrl": "assets/live/rocket.json", // gift special effect address, if it start
"giftName": "Super Rocket", // gift name
},
}
};

// The gift does not have special effects (it will slide in on the left side of the
final customInfoPlane = {
    "version": 1.0, // protocol version number
    "businessID": "flutter_live_kit", // Business ID field
    "data": {
      "cmd":
          "send_gift_message", // must be send_gift_message
      "cmdInfo": {
        "type": 2, // gift type
        "giftUrl": "", // URL of gift image
        "giftCount": 1, // number of gifts
        "giftName": "Airplane", // gift name
      },
    }
};

// Ordinary gift (the gift will be displayed in the barrage, and will not slide in
final normalGift = {
    "version": 1.0, // protocol version number
    "businessID": "flutter_live_kit", // business ID field
    "data": {
      "cmd":
          "send_gift_message", // must be send_gift_message
      "cmdInfo": {
        "type": 1, //Ordinary gift
        "giftUrl": "", // URL of gift image
        "giftCount": 1, // number of gifts
        "giftName": "flower", // gift name
        "giftUnits": "Duo", // gift units
      },
    }
};
```

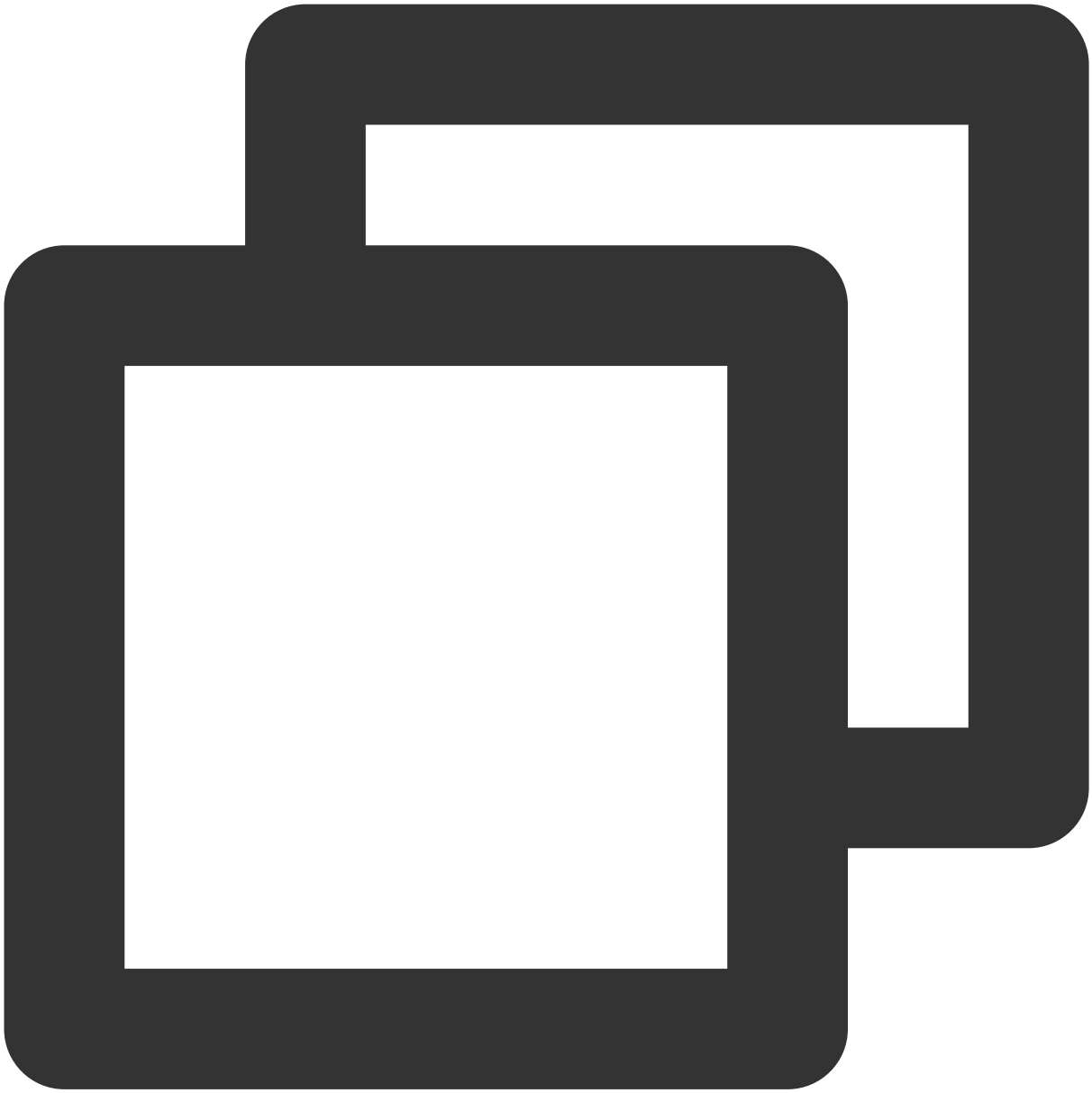
5: Theme

In addition to customization, this plugin also provides the capability of `theme`. The theme is divided into two parts: `color` and `font`. Theme can be customized according to your needs.

API Docs

TencentCloudAvChatRoomData

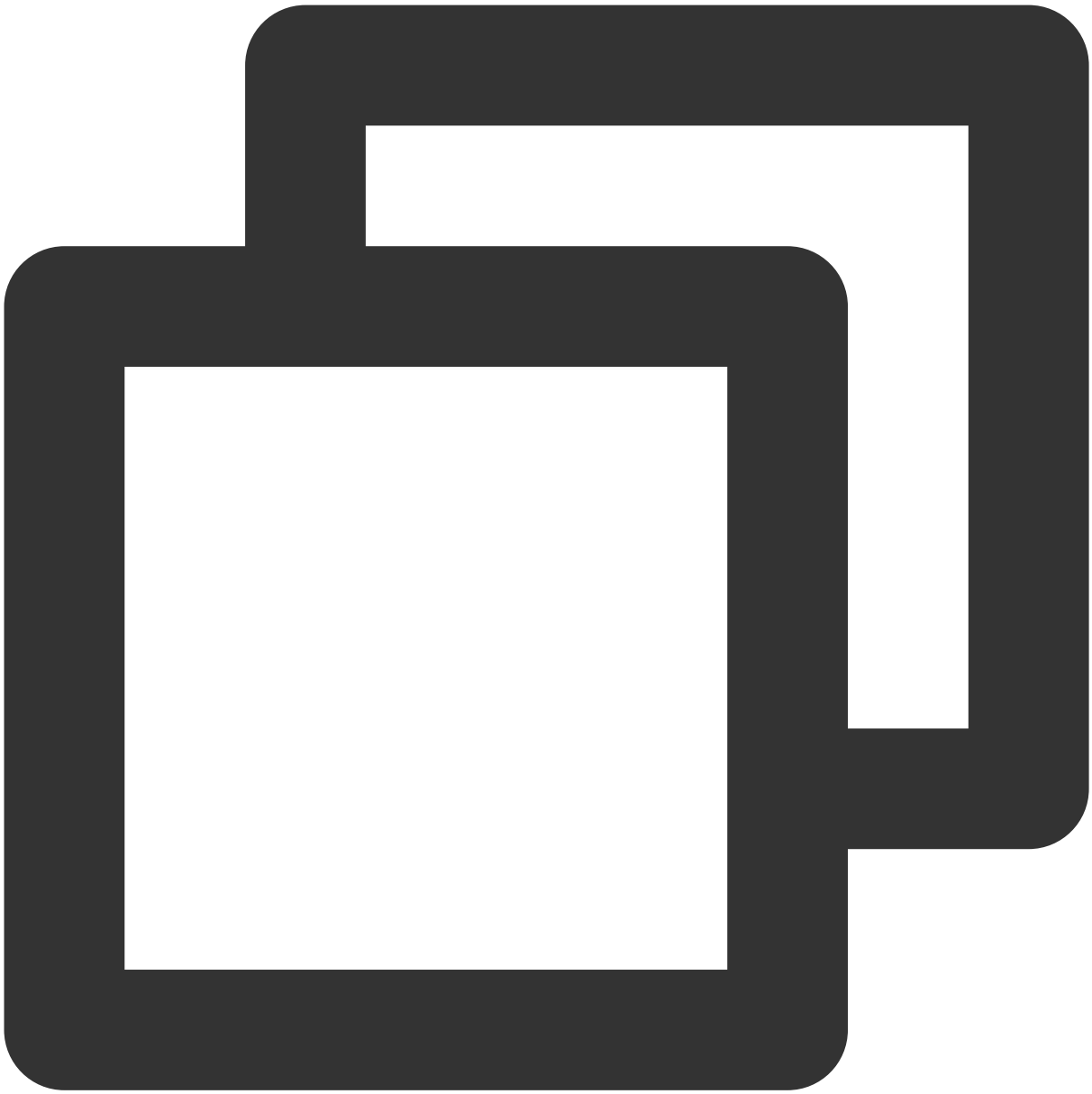
The data that the component needs to use, such as anchor information, broadcast room announcements, etc.



```
TencentCloudAvChatRoomData(
anchorInfo: AnchorInfo(), // anchor information
isSubscribe: false, // whether to subscribe
notification: "Live Room Announcement" // Live Room Announcement
)
```

TencentCloudAvChatRoomConfig

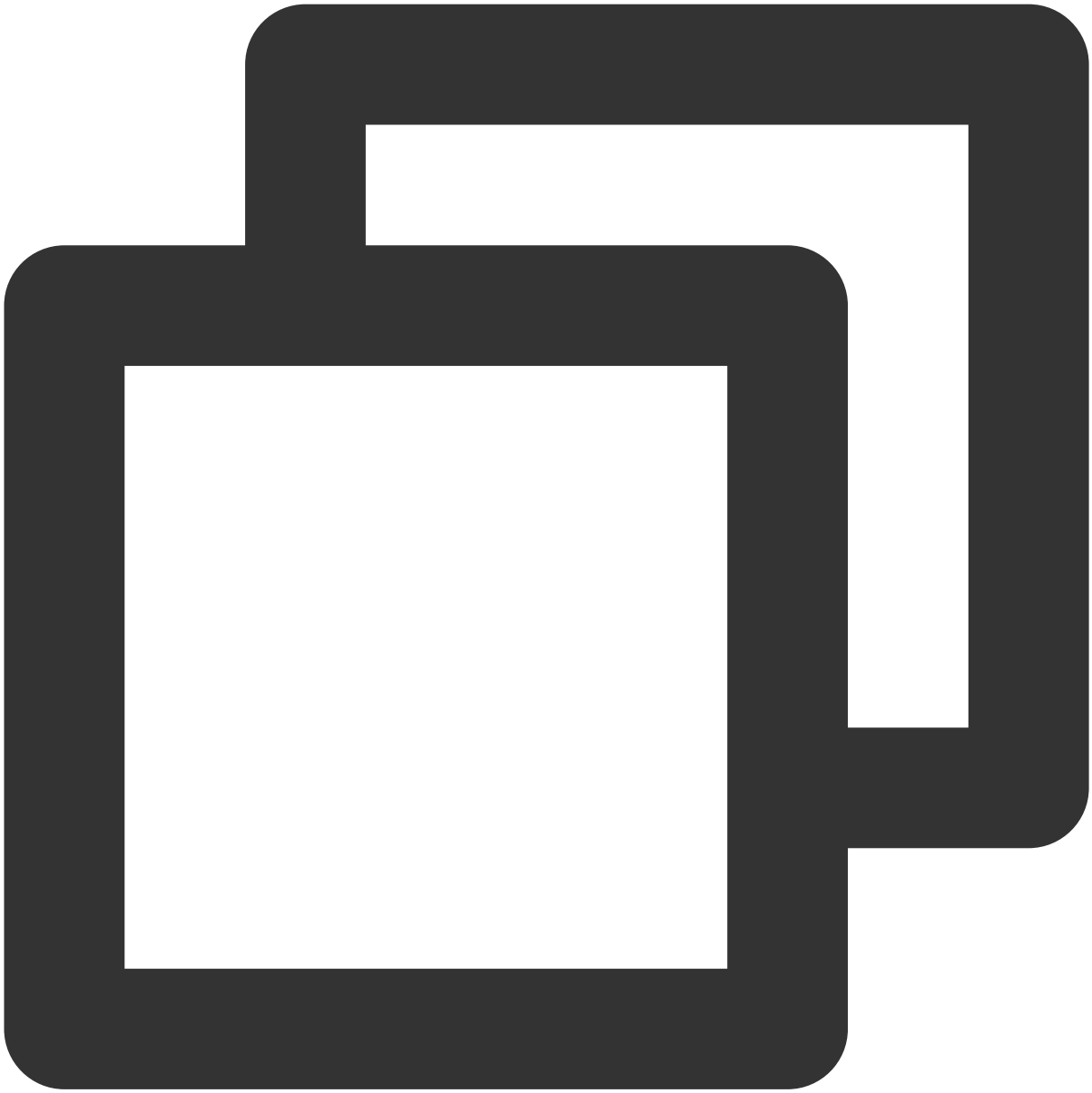
Component configuration information.



```
TencentCloudAvChatRoomConfig(
  avChatRoomID: '', // AV Chat Group. Group ID of AV Chat Room type.[https://cloud.
  loginUserID: '', // login user ID
  sdkAppID: 0, // IM application ID
  userSig: '', // sig generated by user ID and secretKey
  barrageMaxCount: 200, // The maximum number of barrage. The default is 200. When
  giftHttpBase: '', // Gift message http base.
  displayConfig: DisplayConfig() // Can control the display and hiding of some comp
  )
```

TencentCloudAvChatRoomController

Component controller, which can be called outside the component to update data, send messages, etc.



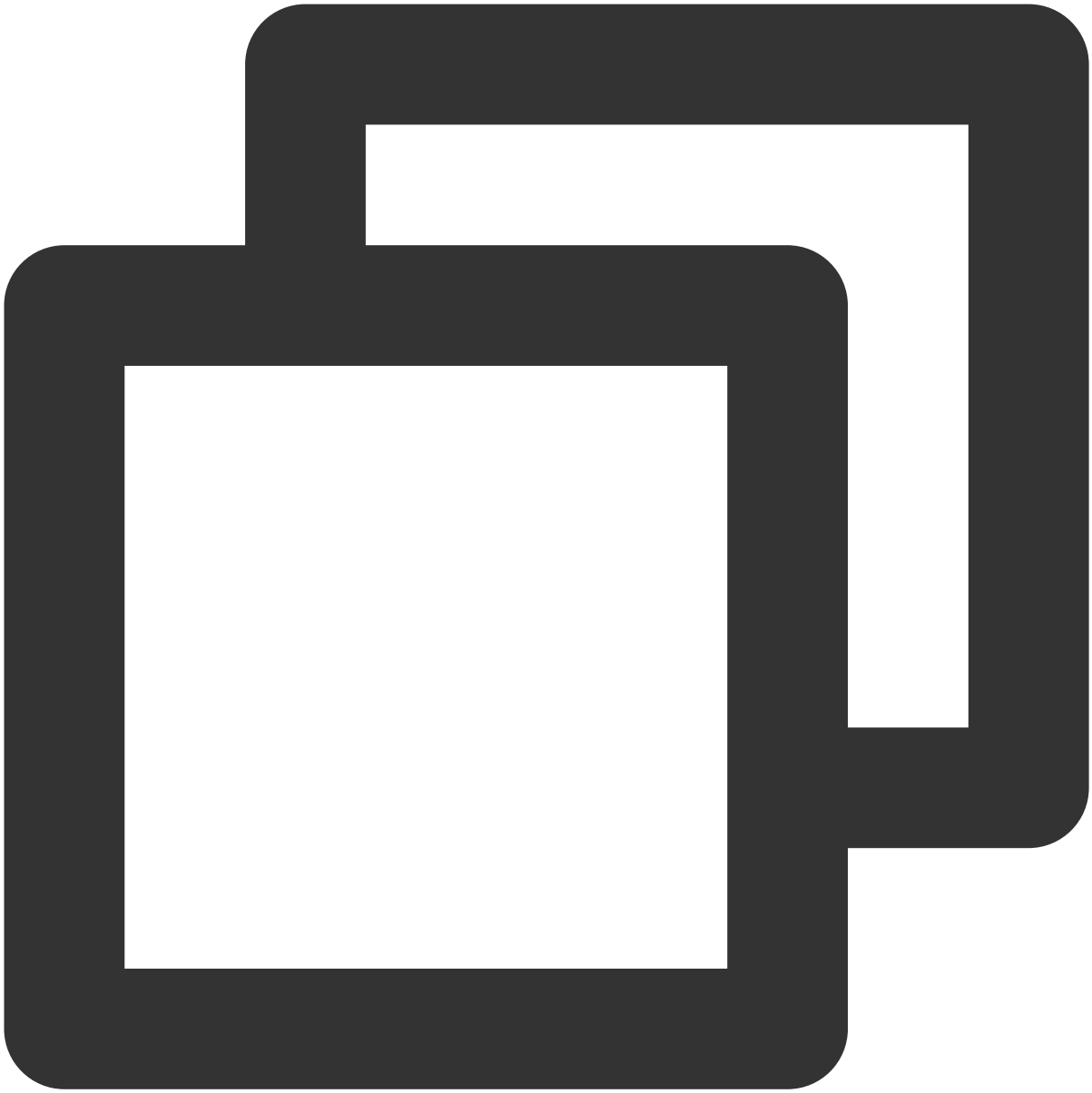
```
final controller = TencentCloudAvChatRoomController();
final _needUpdateData = TencentCloudAvChatRoomData(
        anchorInfo: AnchorInfo(), // anchor information
isSubscribe: false, // whether to subscribe
notification: "Live Room Announcement" // Live Room Announcement
);
final _textString = "I am a text message";
```

```
final customInfoRocket = {
"version": 1.0, // protocol version number
"businessID": "flutter_live_kit", // Business ID field
"data": {
"cmd":
"send_gift_message", // command
"cmdInfo": { // Information carried by the command
"type": 3, // gift type
"giftUrl": "1e8913f8c6d804972887fc179fa1fbd7.png", // gift image address
"giftCount": 1, // number of gifts
"giftSEUrl": "assets/live/rocket.json", // gift special effect address
"giftName": "Super Rocket", // gift name
},
}
};

final customInfoPlane = {
    "version": 1.0,
    "businessID": "flutter_live_kit",
    "data": {
      "cmd":
          "send_gift_message",
      "cmdInfo": {
        "type": 2,
        "giftUrl": "5e175b792cd652016aa87327b278402b.png",
        "giftCount": 1,
        "giftName": "Airplane",
      },
    }
};

final customInfoFlower = {
    "version": 1.0,
    "businessID": "flutter_live_kit",
    "data": {
      "cmd":
          "send_gift_message",
      "cmdInfo": {
        "type": 1,
        "giftUrl": "8f25a2cdeae92538b1e0e8a04f86841a.png",
        "giftCount": 1,
        "giftName": "Flower",
        "giftUnits": "Duo",
      },
    }
};
```

```
// Update the data information of the incoming component.
controller. updateData(_needUpdateData);

// send text message
controller. sendTextMessage(_textString);

// Send a gift message, the gift message needs to follow the specific format above.
controller.sendGiftMessage(jsonEncode(customInfoFlower));

// Send any type of message, [message] needs to be created by yourself
controller. sendMessage(message);

// Play special effects animation (Lottie, SVGA).
controller.playAnimation("assets/live/rocket.json"):
```

TencentCloudAvChatRoomCallback

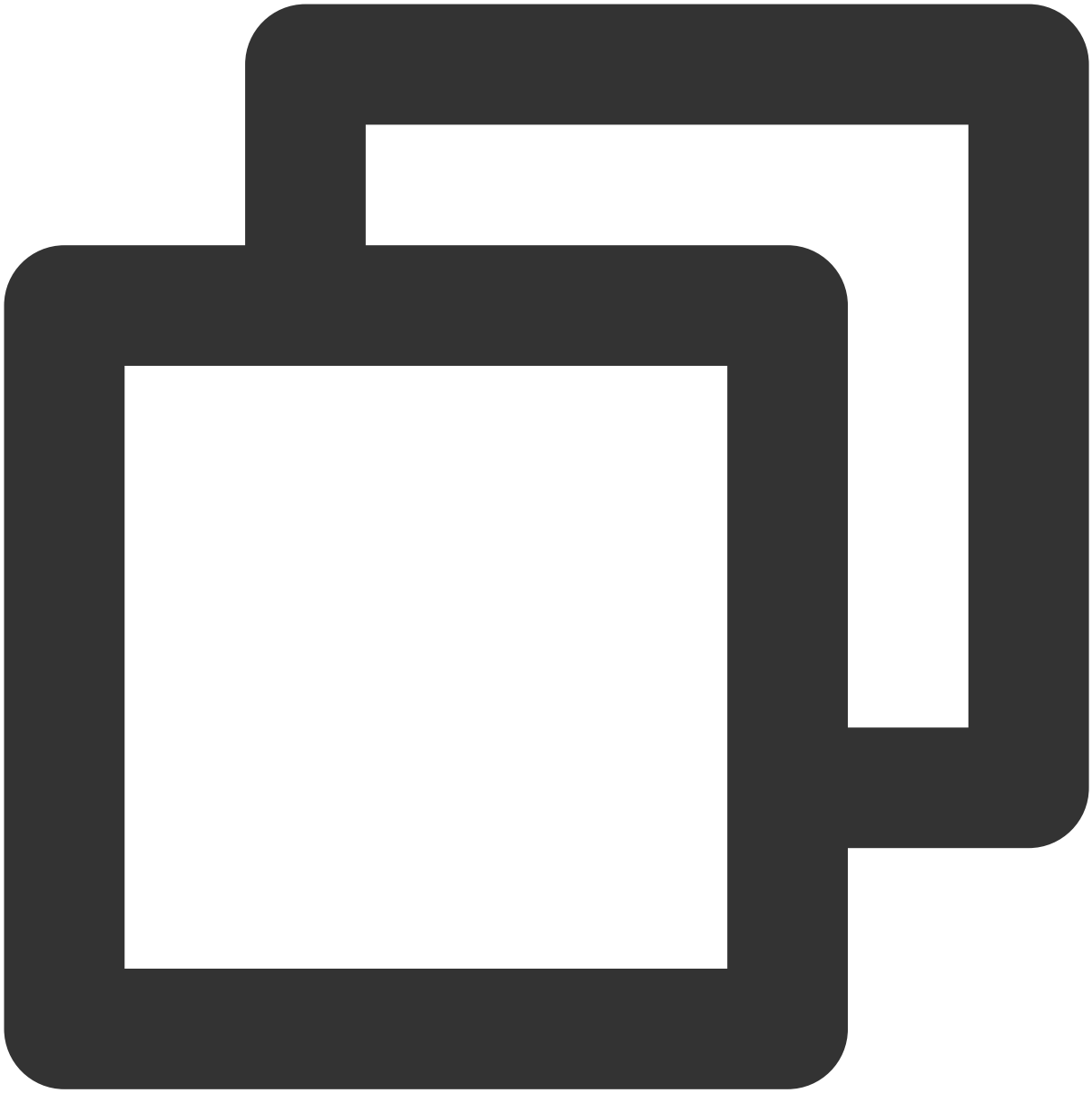
Event callback。



```
TencentCloudAvChatRoomCallback(
onMemberEnter: (memberInfo) {}, // someone enters the live room
onRecvNewMessage: (message) {} // received barrage message
)
```

TencentCloudAvChatRoomCustomWidgets

custom components

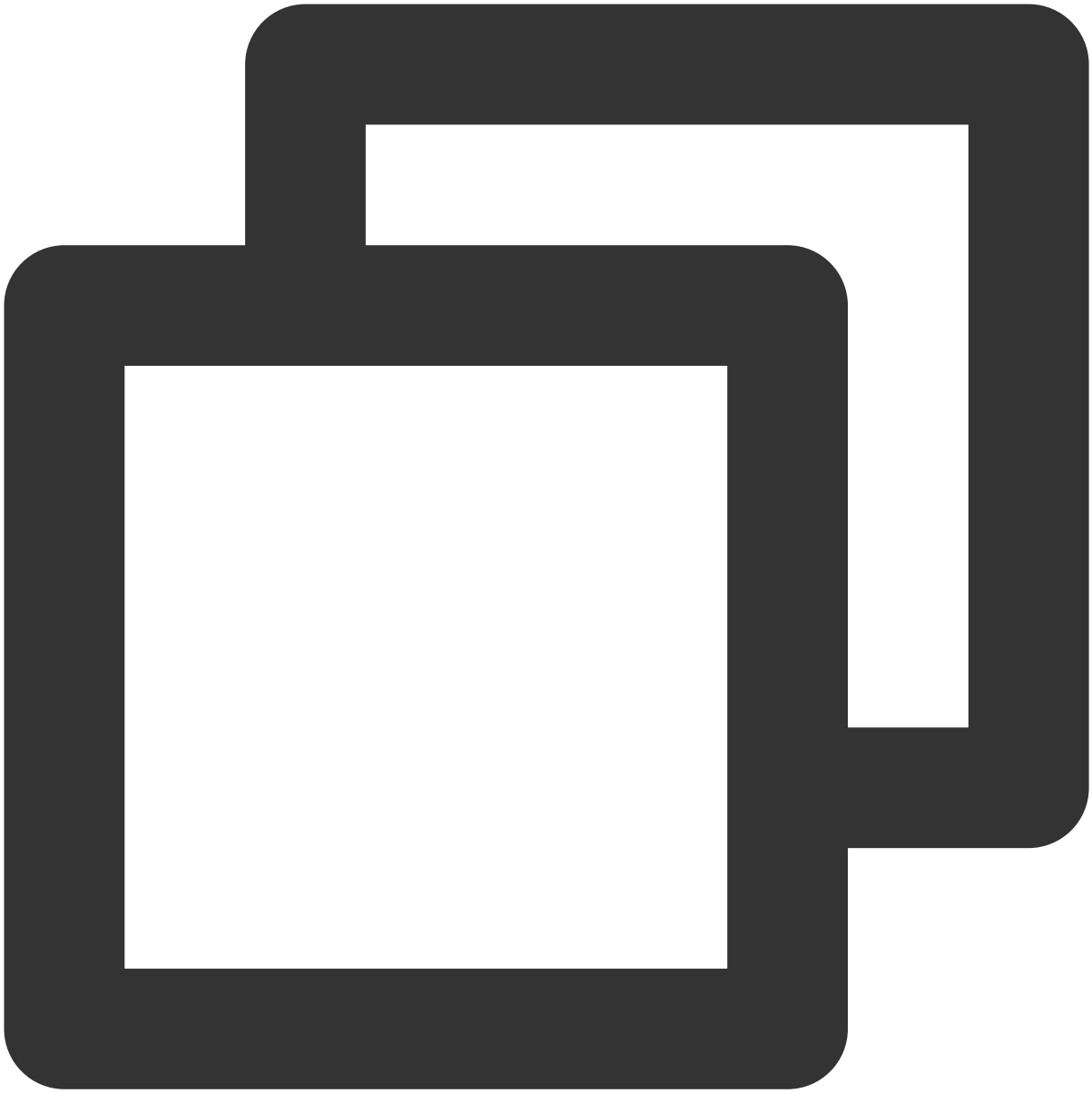


```
TencentCloudAvChatRoomCustomWidgets(
roomHeaderAction: Container(), // The area custom component is displayed in the upp
roomHeaderLeading: Container(), // The area custom component is displayed in the up
roomHeaderTag: Container(), // Below roomHeaderAction and roomHeaderLeading, it is
onlineMemberListPanelBuilder: (context, id) { // Customize the panel that expands a
return Container();
},
anchorInfoPanelBuilder: (context, id) { // Customize the expanded panel after the a
return Container();
},
giftsPanelBuilder: (context) { // Customize the panel displayed after clicking the
```

```
return Container();
},
messageItemBuilder: (context, message, child) { // Customize bullet chat messages
return Container();
},
messageItemPrefixBuilder: (context, message) { // Customize the prefix of bullet ch
return Container();
},
giftMessageBuilder: (context, message) { // Custom gift message, gift message that
return Container();
},
textFieldActionBuilder: (// Customize the lower right area of the screen
context,
) {
return [Container()];
},
textFieldDecoratorBuilder: (context) { // Customize the input box at the bottom lef
return Container();
}
)
```

TencentCloudAvChatRoomTheme

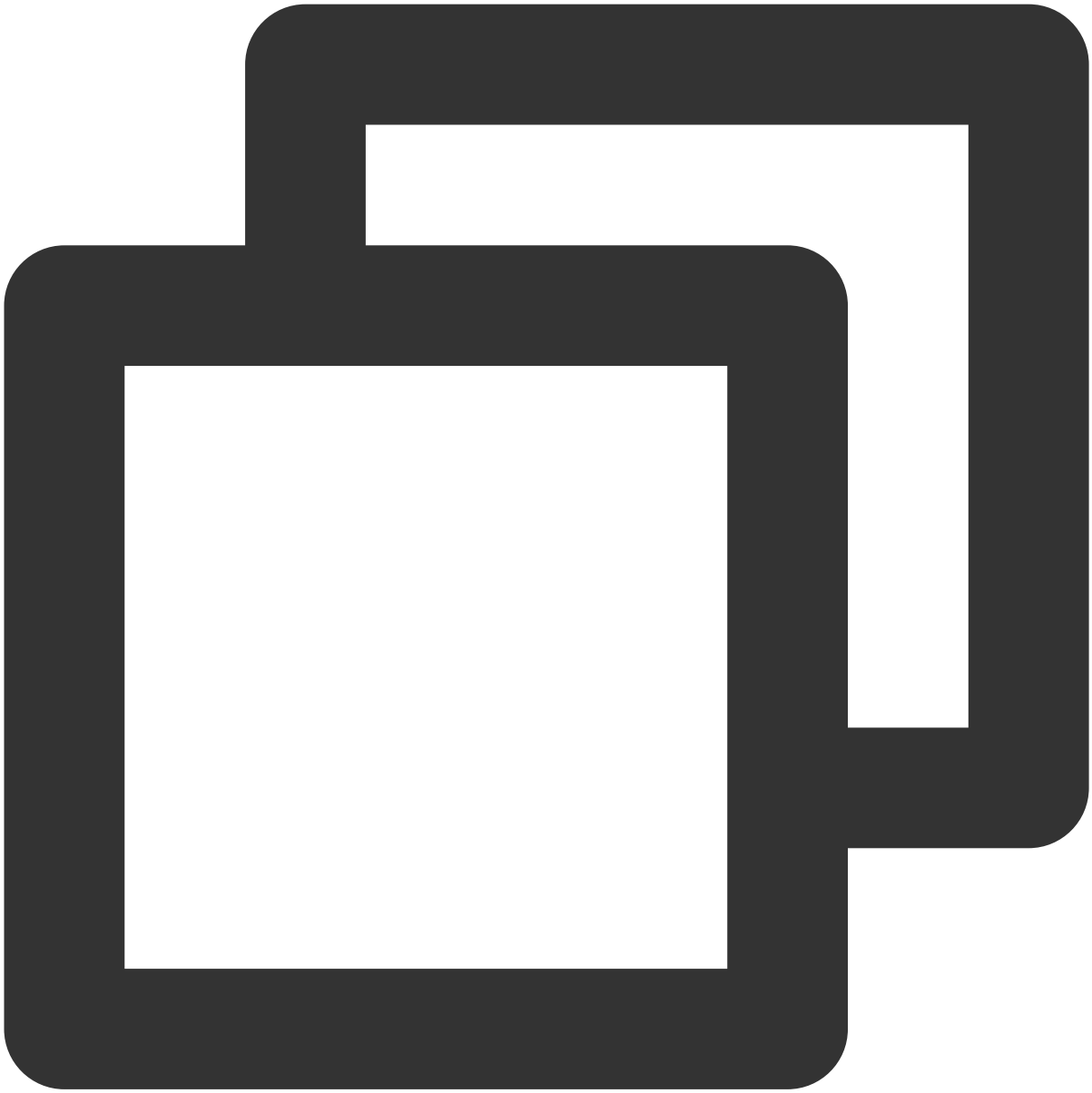
Theme



```
TencentCloudAvChatRoomTheme(
backgroundColor: Colors.black, // The background color of the widget,
hintColor: Colors.red, // hint text color
highlightColor: Colors.orange, // highlight color
accentColor: Colors.white, // foreground color
textTheme: TencentCloudAvChatRoomTextTheme(), // font theme
secondaryColor: Colors.grey, // secondary color
inputDecorationTheme: InputDecorationTheme() // input box theme
)
```

TencentCloudAvChatRoomTextTheme

font theme



```
TencentCloudAvChatRoomTextTheme(
giftBannerSubTitleStyle: TextStyle(), // gift message drawn on the left side of the
giftBannerTitleStyle: TextStyle(), // The title font theme of the gift message draw
anchorTitleStyle: TextStyle(), // Anchor name font theme
anchorSubTitleStyle: TextStyle(), // like font theme
barrageTitleStyle: TextStyle(), // barrage message sender name subject
barrageTextStyle: TextStyle() // Barrage message content theme
)
```

contact us

If there's anything unclear or you have more ideas, feel free to contact us!

Telegram Group

WhatsApp Group

AI Chatbot

최종 업데이트 날짜: : 2024-02-07 17:30:51

With the global popularity of ChatGPT, artificial intelligence (AI) has become the focus of developers today, and mainstream vendors in China have launched their own big model (BM) applications and products. Many vendors have combined their applications with AI to discover new opportunities. The powerful conversational communication capabilities of next-generation large language models (LLMs) are naturally compatible with all kinds of instant messaging scenarios, which brings broad imagination space for the combination of Tencent Cloud Chat and AI.

In office scenarios, users can chat with conversational AI to efficiently make work notes, write documents, collect information, and more. In customer service scenarios, AI-powered smart customer service can provide a conversational experience similar to human customer service and guide users to purchase and use products more effectively. In social scenarios, AI chatbots can provide users with 24-hour online psychological counseling and emotional companionship, increasing user engagement and more. Tencent Cloud Chat, the world's leading provider of communication cloud services, has also seen the huge potential of AI in the instant messaging scenario, and quickly released AI capability call APIs. Based on the communication base provided by Tencent Cloud Chat, developers can freely call industry-leading BM capabilities and empower themselves with rich AI capabilities to efficiently implement scenario-specific innovations.

This document describes how to integrate AI service capabilities into Tencent Cloud Chat through the webhook feature of Chat to build an AI chatbot for users to implement features such as intelligent customer service, creative assistance, and work assistant. (The procedure in this document takes the MiniMax LLM as an example. You can use the same method to integrate other ChatGPT-like services.)

Preparations

Creating a Tencent Cloud Chat account

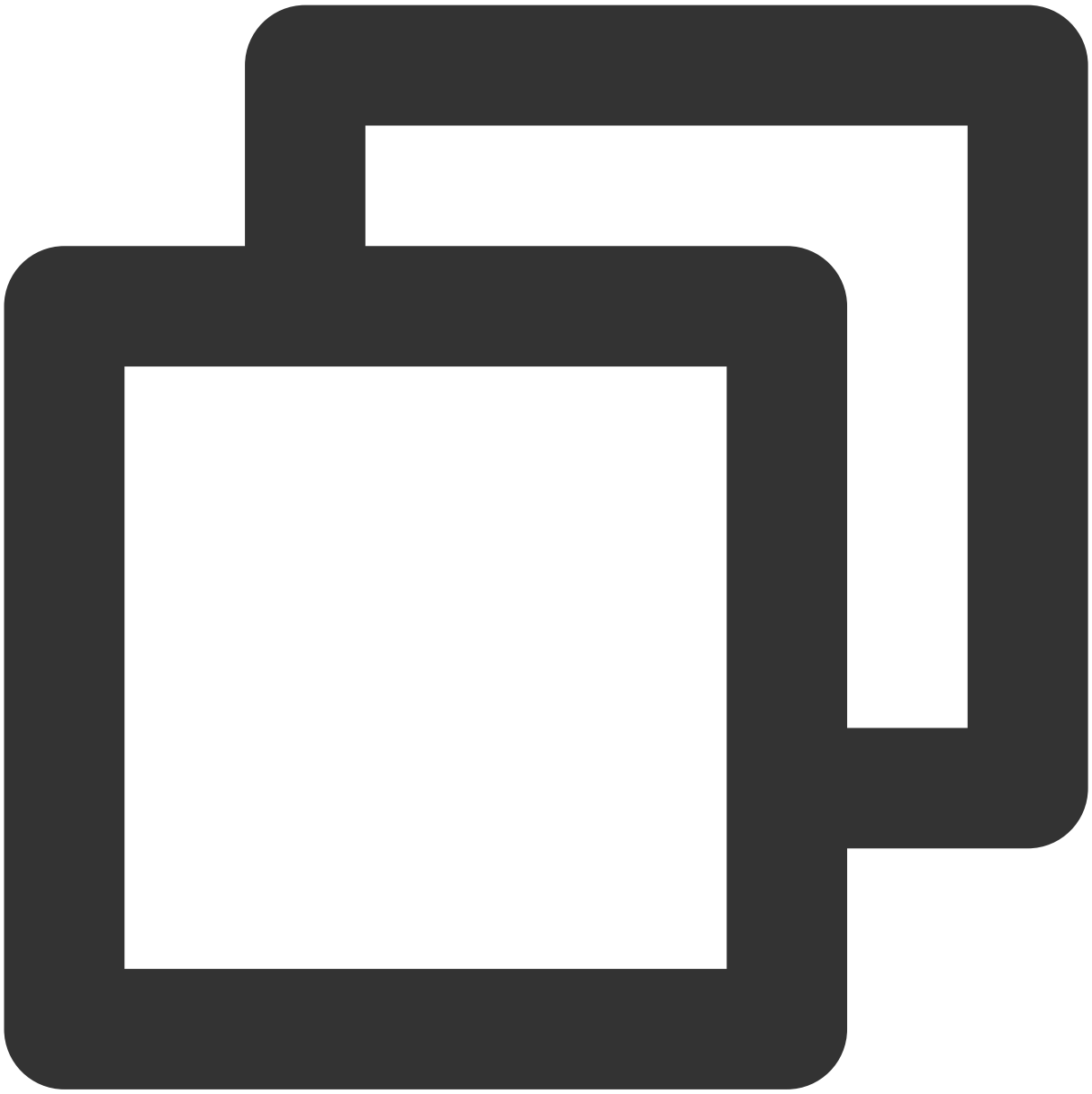
Log in to your Tencent Cloud account, go to the Tencent Cloud Chat console, create an application, get the application's SDKAppID and key (Tencent Cloud Chat key), and create an admin account `administrator`.

Signing up for an account with the corresponding AI service provider

Sign up for an account with the provider of the AI service to be integrated, log in, and get the API key (`AI_SECRET_KEY`).

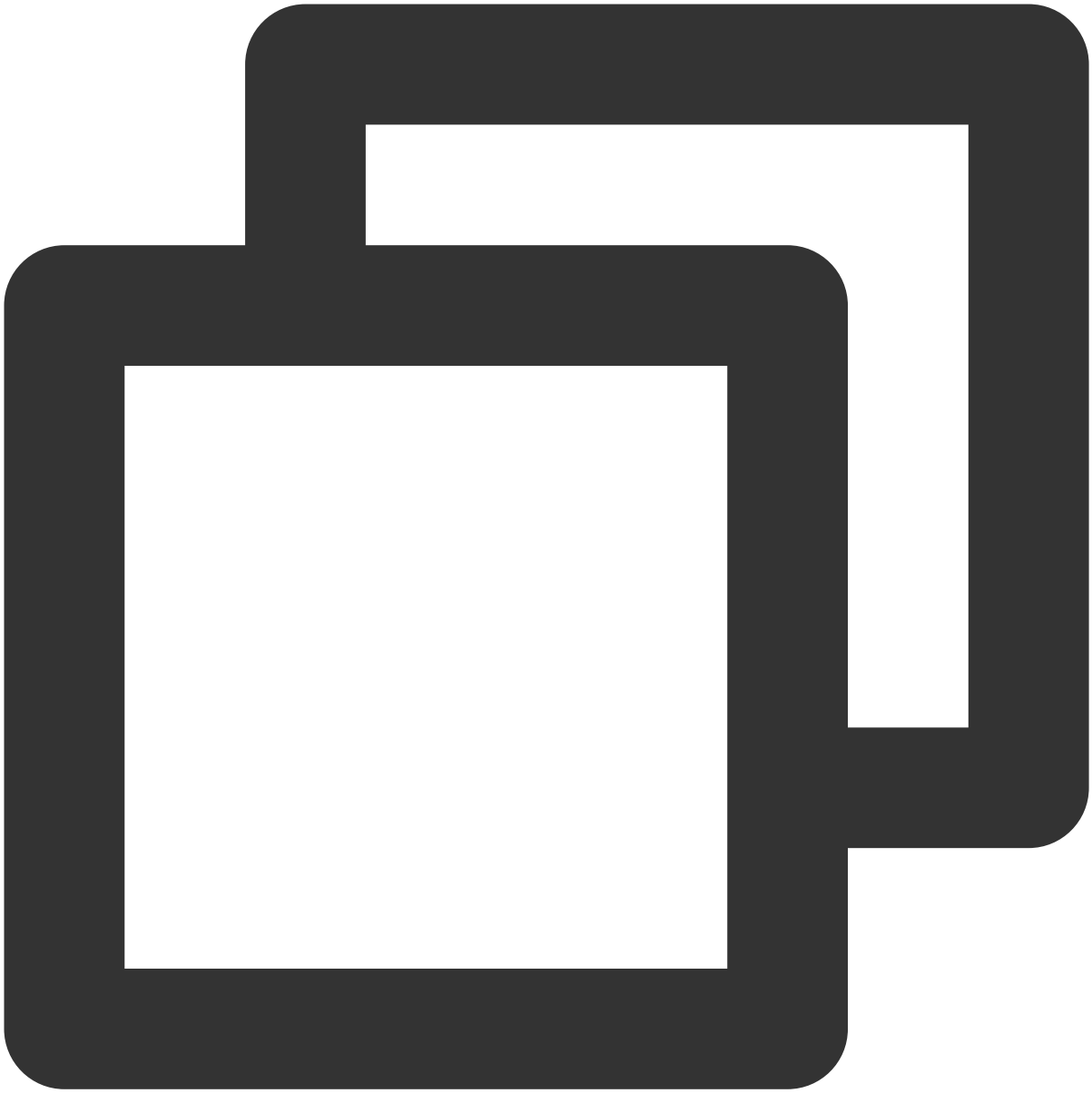
Creating a Tencent Cloud Chat chatbot account

Create a Tencent Cloud Chat chatbot account through the RESTful API. The Tencent Cloud Chat chatbot is a special user whose user ID begins with `@RBT#` .



```
curl -d '{"UserID":"@RBT#001","Nick":"MyRobot"}' "https://console.tim.qq.com/v4/ope
```

Replace `sdkappid={}` and `usersig={}` in the command above with your SDKAppID and the UserSig generated based on the Tencent Cloud Chat key. For more information, see Generating UserSig. After you run the command in Linux, the Tencent Cloud server returns the following information:



```
{"ActionStatus": "OK", "ErrorCode": 0, "ErrorInfo": ""}
```

The information above indicates that the chatbot `@RBT#001` with the nickname `MyRobot` was created successfully.

Configuring Tencent Cloud Chat webhooks

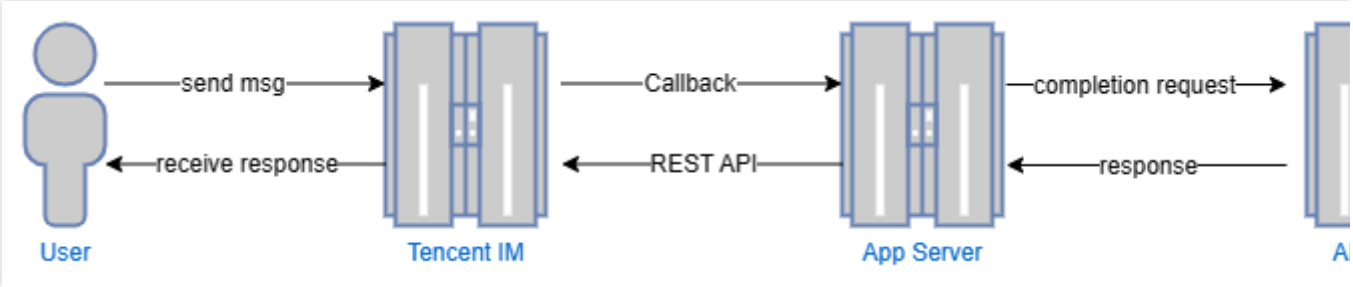
A Tencent Cloud Chat webhook is a request sent by the Tencent Cloud Chat backend to the backend server of the corresponding application before or after an event. The application backend can then perform the necessary data synchronization or intervene in the subsequent processing of the event. We will use a "robot event webhook" to listen

for and react to user messages sent to the chatbot or @RBT# events in group chats. You need to locate and click "Robot Event Webhook" in the Tencent Cloud Chat console to enable the feature and save the settings.

Writing the Application Backend Service

Taking a one-to-one chat as an example, the overall working process is as follows:

1. The `user1` user sends the "hello" message to the chatbot `@RBT#001` .
2. The Tencent Cloud Chat backend sends a webhook to notify the application backend of the event.
3. The application backend receives the event notification which contains information such as the message sender `user1` , message recipient `@RBT#001` , and message content `hello` .
4. The application backend calls the AI service API (MiniMax API) and receives the response containing the reply message, such as "nice to meet you".
5. The application backend calls the Tencent Cloud Chat RESTful API (API `sendmsg` for a one-to-one chat and API `send_group_msg` for a group chat) to send the reply message to `user1` as `@RBT#001` .



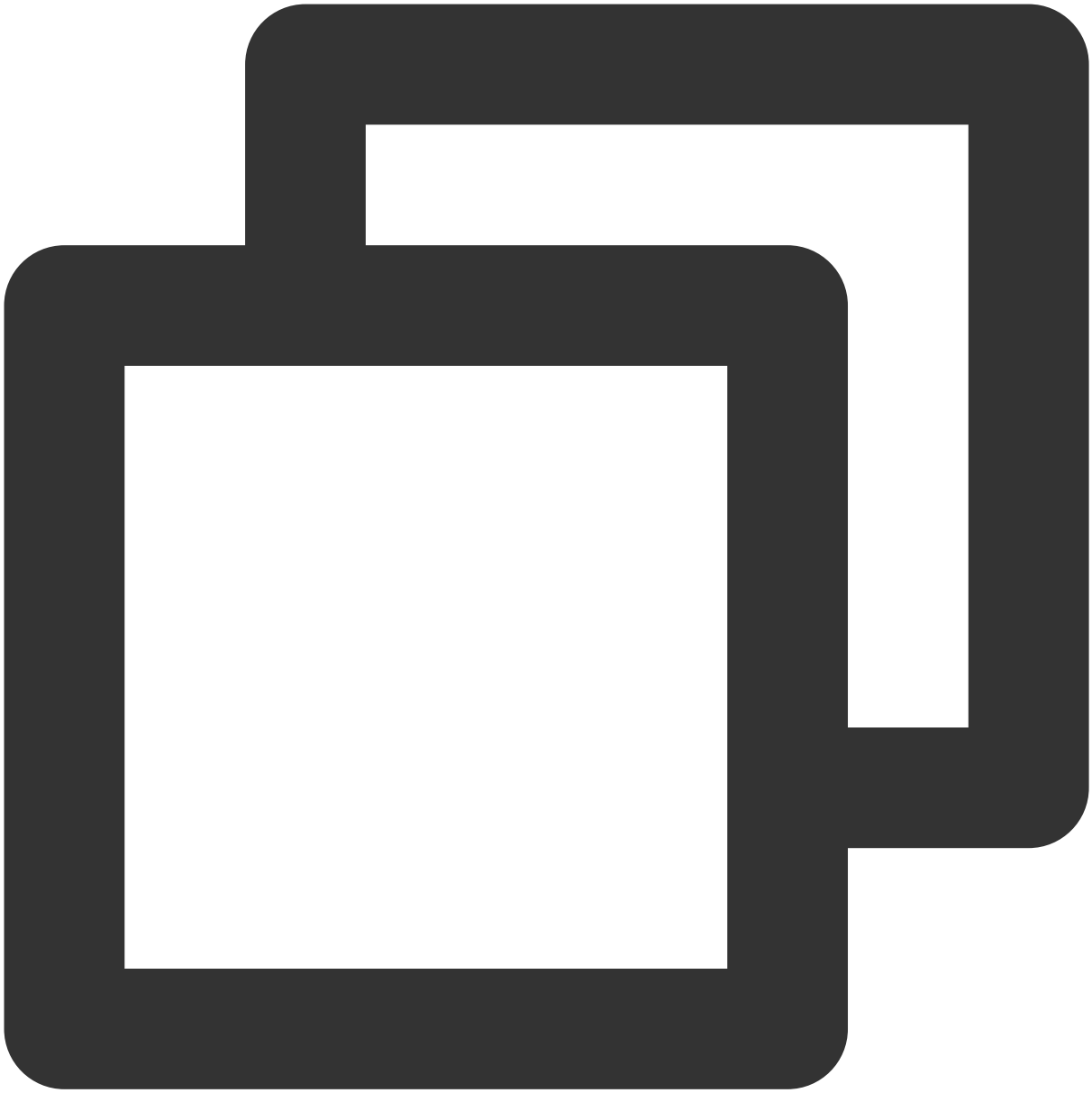
Take the Go programming language as an example, the key code of the application backend is as follows.

Note:

The following code is for demonstration only and omits a lot of exception handling code. It cannot be used directly in production environments.

Distributing and processing the webhook command

We create an HTTP service which is listened to on port 80 and register a handler with the `/im` URL that handles all requests sent to `http://im` . All webhook requests sent by Tencent Cloud Chat contain a `CallbackCommand` parameter, with different values representing different webhook commands. The handler performs processing according to the `CallbackCommand` parameter set by Tencent Cloud Chat.

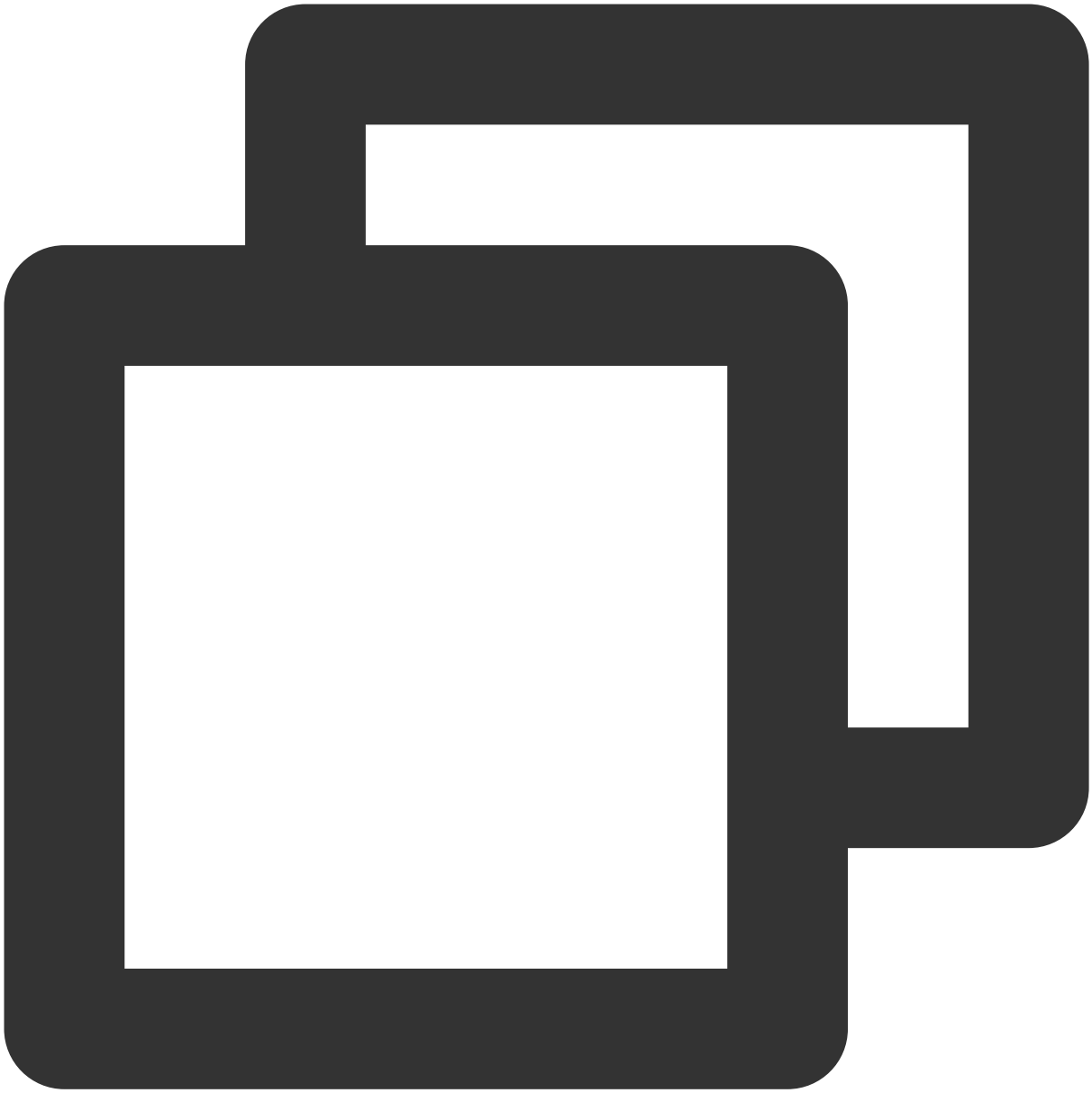


```
func handler(w http.ResponseWriter, r *http.Request) {
   command := r.URL.Query().Get("CallbackCommand")
   reqbody, _ := io.ReadAll(r.Body)
   var rspbody []byte
   switch command {
   case "Bot.OnC2CMessage": // Chatbot's webhook command word for a one-to-one mess
       dealC2c(context.Background(), reqbody)
       rspbody = []byte("{\\"ActionStatus\\": \\"OK\\", \\"ErrorCode\\": 0, \\"Erro
   default:
       rspbody = []byte("invalid CallbackCommand.")
   }
```

```
    w.Write(rspbody)
}

func main() {  // Register a handler to process the webhook command sent to the app
    http.HandleFunc("/im", handler)
    http.ListenAndServe(":80", nil)
}
```

Processing the one-to-one message received by the chatbot

When processing a one-to-one message, we first check that the sender is not a chatbot (generally a chatbot does not send a message to another chatbot) to prevent infinite webhook loops. We then parse the body of the message to obtain the content text of the message sent by the user to the chatbot, save the sender's UserID into the context to facilitate the subsequent action of calling the RESTful API to reply, and finally call `askAI` to request the AI service.

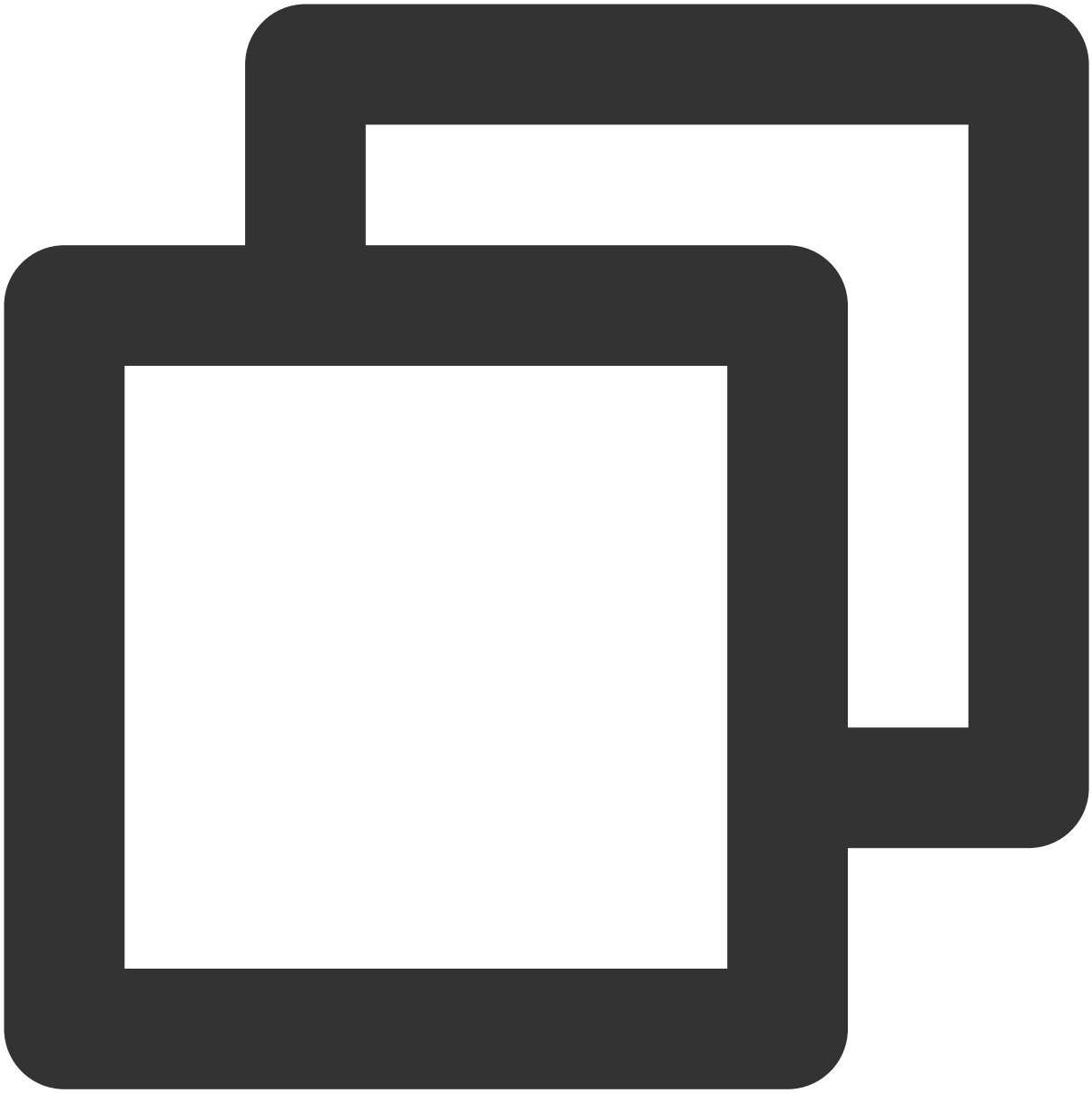


```
func dealC2c(ctx context.Context, reqbody []byte) error {
  root, _ := simplejson.NewJson(reqbody)
  jFromAccount := root.Get("From_Account")
  fromAccount, _ = jFromAccount.String()
  // Check the sender's ID to avoid processing requests sent by one chatbot to anot
  if strings.HasPrefix(fromAccount, "@RBT#") {
    return nil
  }
  jToAccount := root.Get("To_Account")
  toAccount, _ := jToAccount.String()
  msgBodyList, _ := root.Get("MsgBody").Array()
```

```
    for _, m := range msgBodyList {
      msgBody, _ := m.(map[string]interface{})
      msgType, _ := msgBody["MsgType"].(string)
      if msgType != "TIMTextElem" {
        continue
      }
      msgContent, _ := msgBody["MsgContent"].(map[string]interface{})
      text, _ := msgContent["Text"].(string)
      ctx = context.WithValue(ctx, "from", fromAccount)
      ctx = context.WithValue(ctx, "to", toAccount)
      go askAI(ctx, text)
    }
    return nil
  }
```

Calling the AI service API

In this step, we use a third-party AI service, MiniMax LLM, to implement intelligent chat. Any other AI service can be used instead of the MiniMax LLM service. Note that here we demonstrate a simple `completion` API that does not contain the context of the conversation, and you can see the MiniMax documentation for details on other APIs as needed.

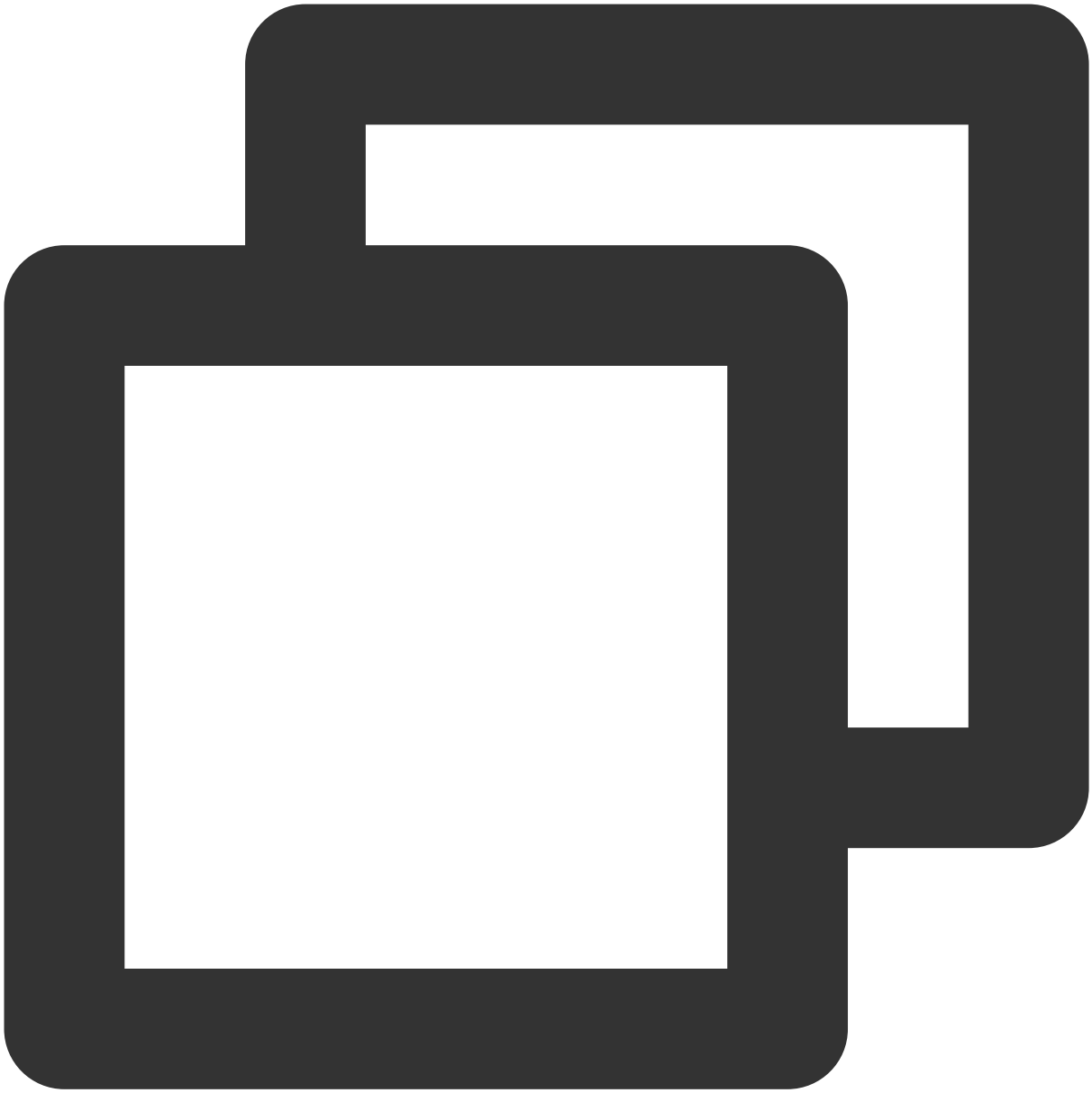


```
type MiniMaxRsp struct {
  Reply string `json:"reply"`
}

// Send a request to MiniMax and get a reply
func askAI(ctx context.Context, prompt string) {
  url := "https://api.minimax.chat/v1/text/completion"
  var reqData = []byte(`{
    "model": "abab5-completion",
    "prompt": prompt
  }`)
```

```
    request, _ := http.NewRequest("POST", url, bytes.NewBuffer(reqData))
    request.Header.Set("Content-Type", "application/json; charset=UTF-8
    request.Header.Set("Authorization", API_SECRET_KEY)
    client := &http.Client{}
    response, _ := client.Do(request)
    defer response.Body.Close()
    body, _ := ioutil.ReadAll(response.Body)
    rsp := &MiniMaxRsp{}
    json.Unmarshal(body, rsp)
    reply(ctx, rsp.Reply) // Send the content replied by the AI service to the user
}
```

Returning the result replied by the AI service to the user

After receiving the reply from the AI service, we only need to call the `sendmsg` RESTful API of Tencent Cloud Chat to simulate the chatbot to reply to the user, specifying the sender of the message as `@RBT#001` and the recipient as `user1` .

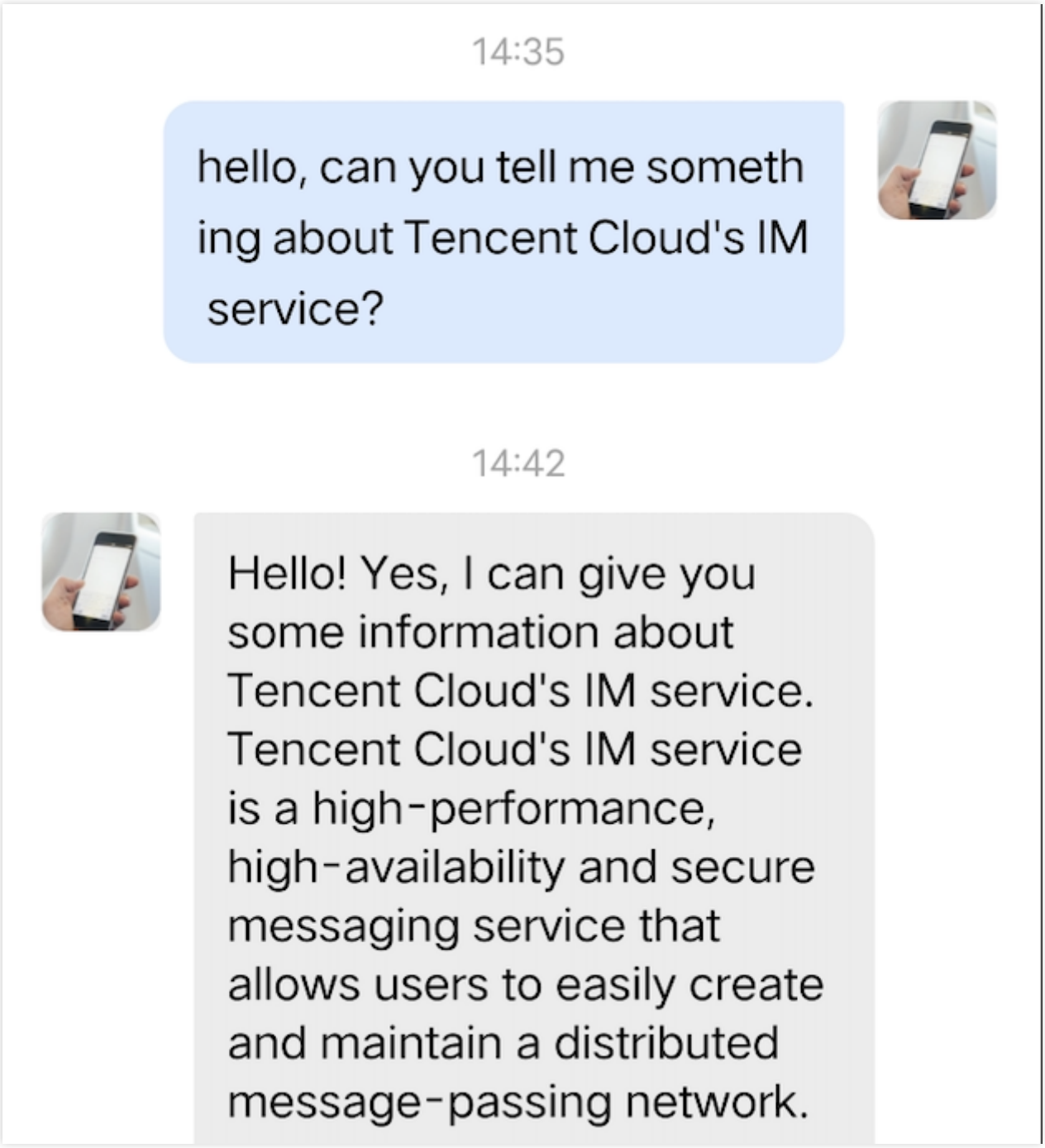


```
// Send a RESTful API request
func doRestAPI(host string, sdkappid int, admin, usersig, command, body string) {
  url := fmt.Sprintf("https://%s/v4/%s?sdkappid=%d&identifier=%s&usersig=%s&random=
    host, command, sdkappid, admin, usersig, rand.Uint32())
  req, _ := http.NewRequest("POST", url, bytes.NewBufferString(body))
  req.Header.Set("Content-Type", "application/json")
  cli := &http.Client{}
  rsp, err := cli.Do(req)
  if err != nil {
    log.Printf("REST API failed. %s", err.Error())
    return
```

```
    }
    defer rsp.Body.Close()
    rsptext, _ := io.ReadAll(rsp.Body)
    log.Printf("rsp:%s", rsptext)
}

// Call the RESTful API of Tencent Cloud Chat to reply to the user
func reply(ctx context.Context, text string) {
    rsp := make(map[string]interface{})
    msgbody := []map[string]interface{}{{
        "MsgType":    "TIMTextElem",
        "MsgContent": map[string]interface{}{"Text": text},
    }}
    // For the implementation of `GenUserSig`, see the Tencent Cloud documentation.
    usersig, _ := GenUserSig(IM_SDKAPPID, IM_KEY, "administrator", 60)
    rsp["From_Account"] = ctx.Value("to").(string) //"@RBT#001"
    rsp["To_Account"] = ctx.Value("from").(string)
    rsp["SyncOtherMachine"] = 2
    rsp["MsgLifeTime"] = 60 * 60 * 24 * 7
    rsp["MsgSeq"] = rand.Uint32()
    rsp["MsgRandom"] = rand.Uint32()
    rsp["MsgBody"] = msgbody
    rspbody, _ := json.Marshal(rsp)
    doRestAPI("console.tim.qq.com", IM_SDKAPPID, "administrator", usersig, "openim/se
}
```

Effect Demonstration

The following demonstrates the final implementation effect of the Tencent Cloud Chat chatbot demo:



With the above steps, we have implemented one-to-one chat connectivity between the Tencent Cloud Chat server side and the MiniMaxAI open platform. It is also possible to integrate AI services from another AI service provider following the above steps by simply replacing the `askAI` function with the corresponding API call from that AI service provider. For group chat chatbots, only the implementation of the `Bot.OnGroupMessage` webhook command processing needs to be supplemented.

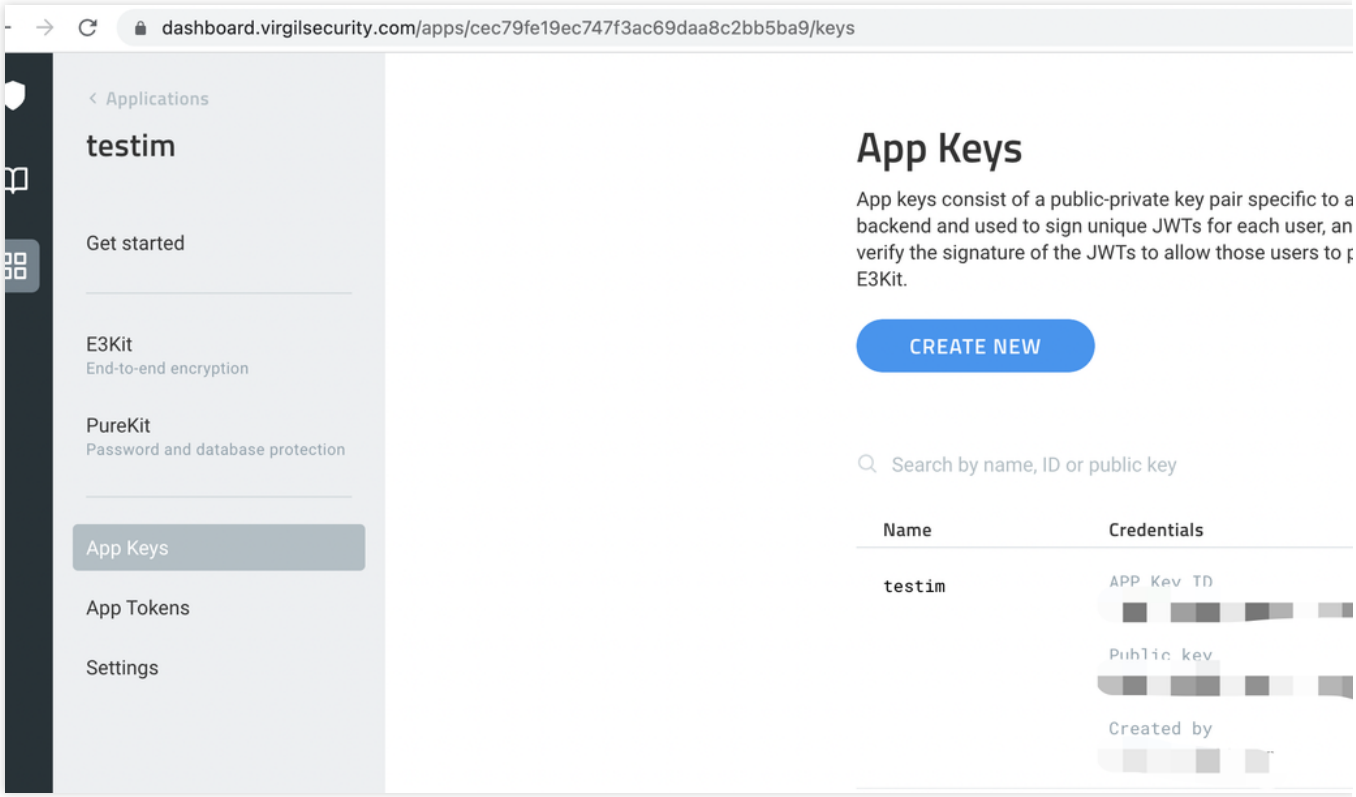
End-to-end encrypted chat with Virgil

최종 업데이트 날짜: : 2024-02-07 17:30:51

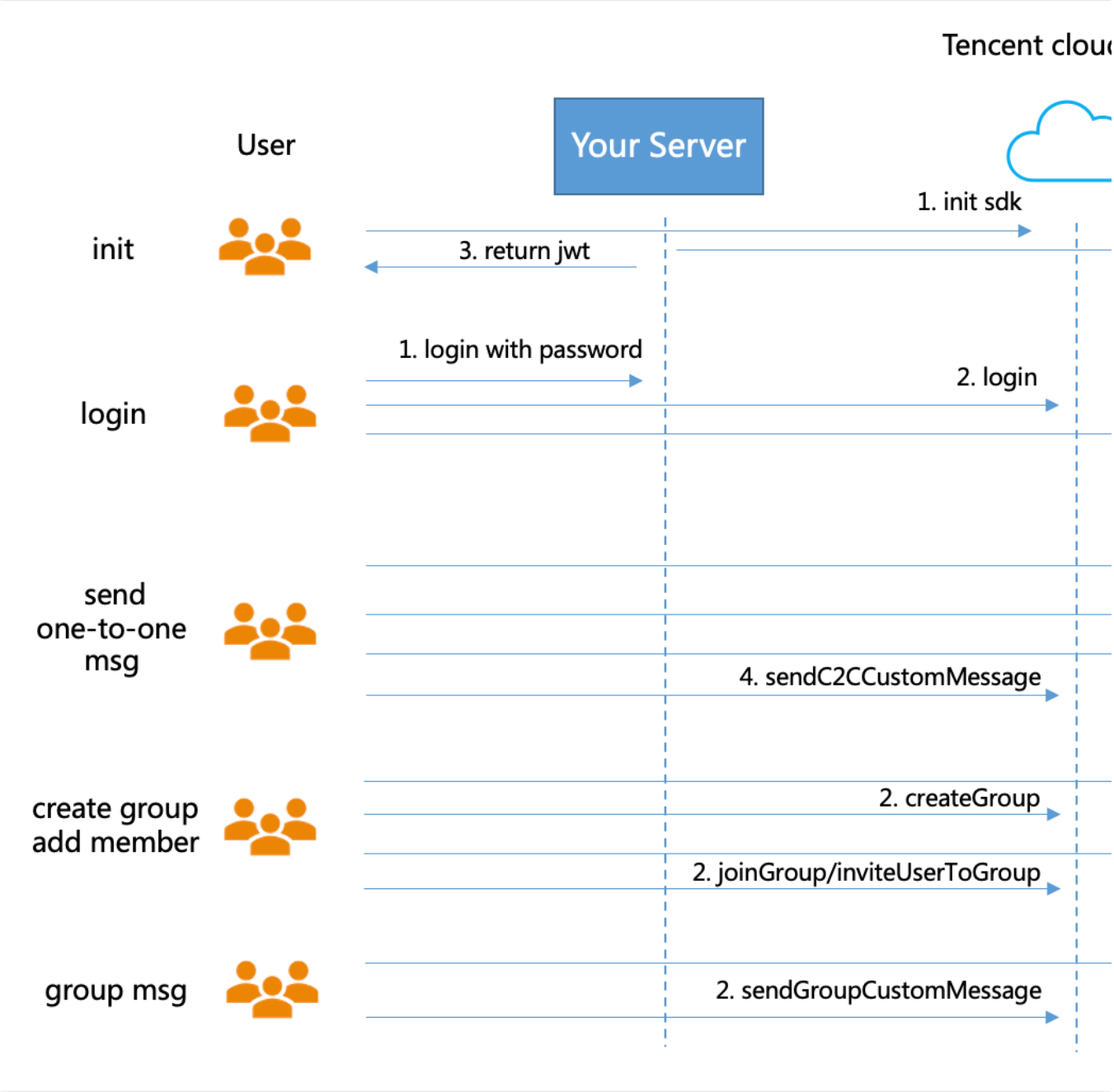
End-to-end encrypted chat with Virgil

Overview

This solution should be implemented by the customer's application based on the E3Kit of virgin security and Tencent Cloud Chat Sdk. The E3Kit document link: Virtual gild E3Kit|Virtual gild Security
Learn about the GroupEncryption-End-to-End-Encryption-E3Kit|Virtual Security documentation before reading the following solutions, especially the JWT token（JSON Web Token）, create channel/create group, encrypt and decrypt messages.
Open application in virgin security console before using. This product is a paid product. See Pricing|Virgin Security for specific pricing.



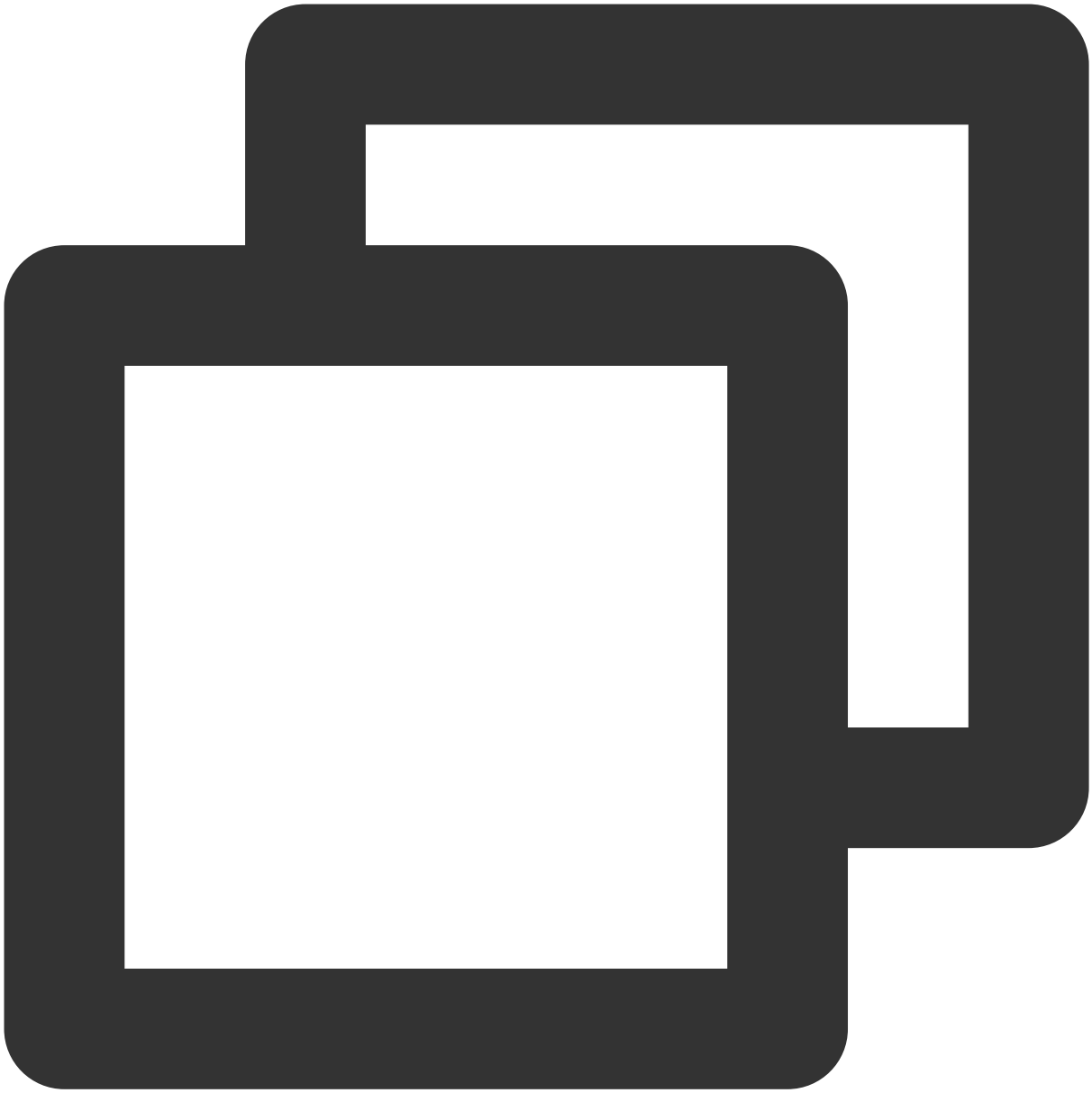
The diagram below shows the high-level communication flow：



Broad implementation overview(example:Android)

Initialization

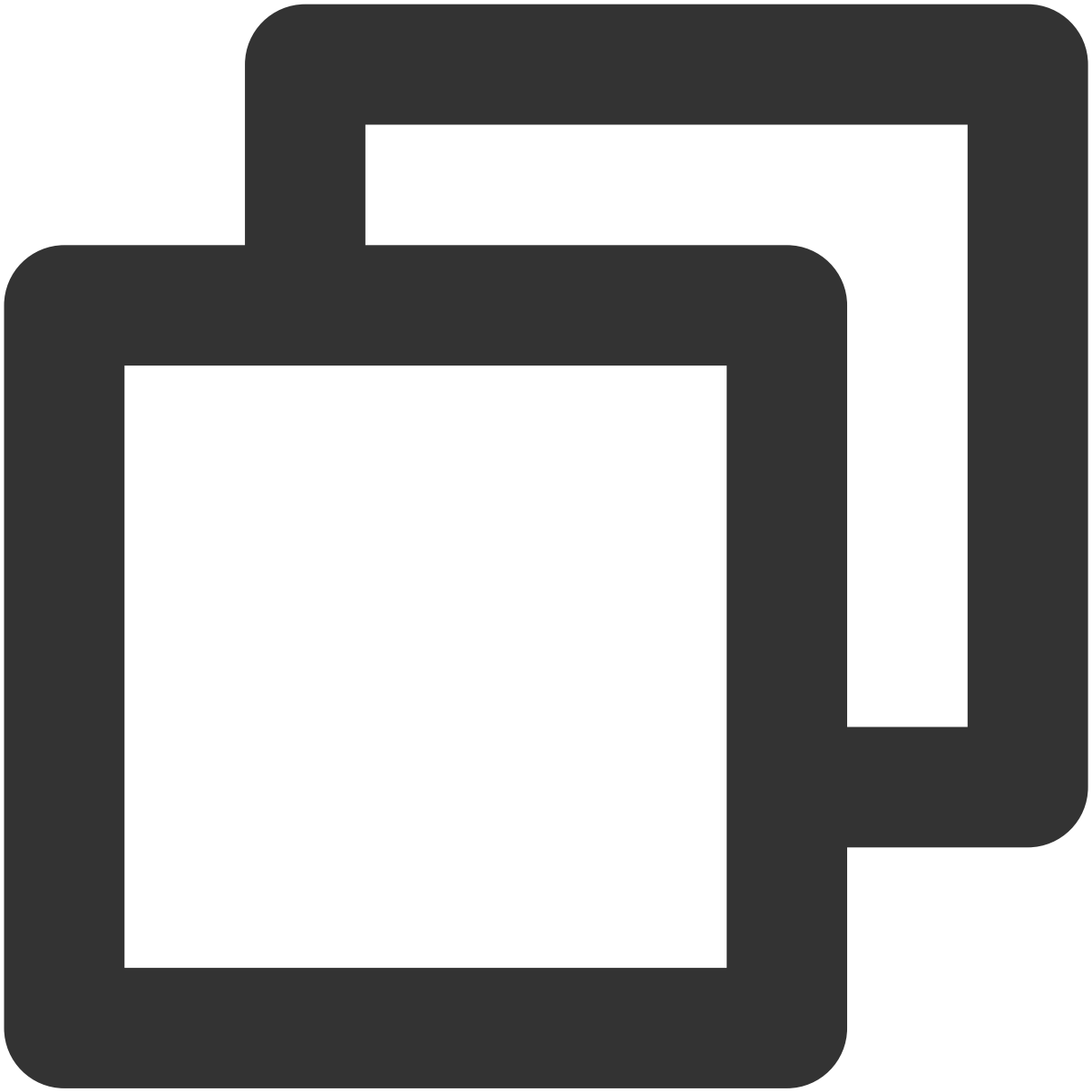
1.Tencent Cloud imsdk initialization



```
// 1. Get the `SDKAppID` from the Chat console.
// 2. Initialize the `config` object.
V2TIMSDKConfig config = new V2TIMSDKConfig();
// 3. Specify the log output level.
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_INFO);
// 4. Add the `V2TIMSDKListener` event listener. `sdkListener` is the implementatio
V2TIMManager.getInstance().addIMSDKListener(sdkListener);
// 5. Initialize the IM SDK. You can call the login API as soon as you call this AP
V2TIMManager.getInstance().initSDK(context, sdkAppID, config);
```

2.E3Kit initialization, passing in the console configuration information, and generating jwt. Corresponding operation document link: GenerateClientTokens-GetStarted-E3Kit | VirgilSecurity

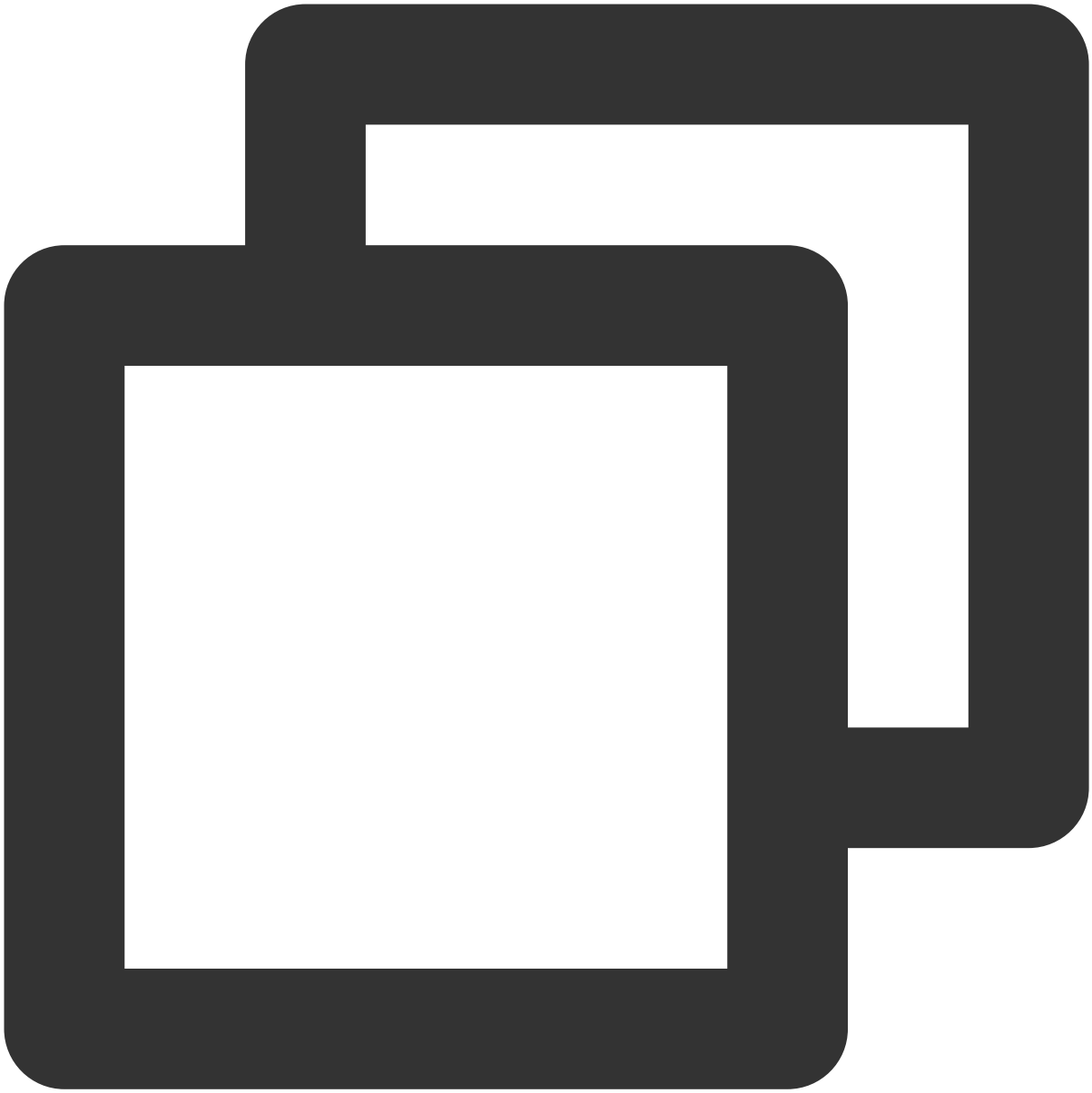
The server generates jwttoken and sends it to the client



```
// generate jwt
// App Key (you got this Key at the Virgil Dashboard)
String appKeyBase64 = "MC4CAQAwBQYDK2VwBCIEINlK4BhgsijAbNmUqU6us0ZU9MGi+HxdYCA6TdZe
byte[] appKeyData = ConvertionUtils.base64ToBytes(appKeyBase64);

// Crypto library imports a key pair
VirgilCrypto crypto = new VirgilCrypto();
```

```
VirgilKeyPair keyPair = crypto.importPrivateKey(appKeyData);

// Initialize an access token signer that signs users JWTs
VirgilAccessTokenSigner accessTokenSigner = new VirgilAccessTokenSigner();

// Use your App Credentials you got at the Virgil Dashboard:
String appId = "be00e10e4e1f4bf58f9b4dc85d79c77a";
String appKeyId = "70b447e321f3a0fd";
TimeSpan ttl = TimeSpan.fromTime(1, TimeUnit.HOURS); // 1 hour - JWT's lifetime

// Setup a JWT generator with the required parameters:
JwtGenerator jwtGenerator =
    new JwtGenerator(appId, keyPair.getPrivateKey(), appKeyId, ttl, accessTokenSign

// Generate a JWT for a user
// Remember that you must provide each user with a unique JWT.
// Each JWT contains unique user's identity (in this case - Alice).
// Identity can be any value: name, email, some id etc.
String identity = "Alice";
Jwt aliceJwt = jwtGenerator.generateToken(identity);

// As a result you get user's JWT, it looks like this: "eyJraWQiOiI3MGI0NDdlMzIxZjN
// You can provide users with JWT at registration or authorization steps.
// Send a JWT to client-side.
String jwtString = aliceJwt.stringRepresentation();
```

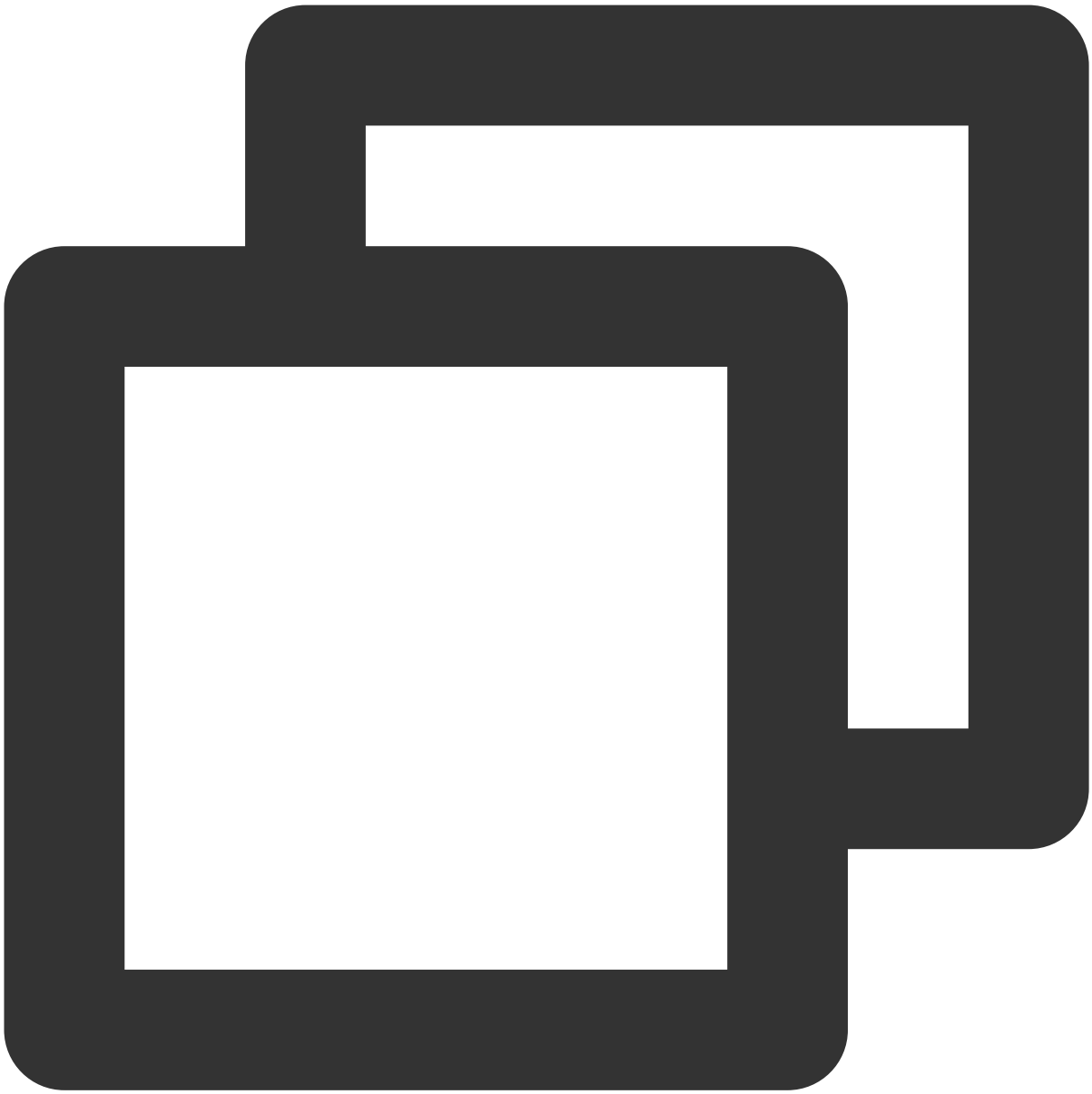
The Android client initializes the E3Kit, tokenCallback is the jwttoken returned by the above server, and User1 is the user ID



```
// initialization E3Kit
// create EThreeParams with mandatory parameters
// such as identity, tokenCallback and context
EThreeParams params = new EThreeParams("User1",
   tokenCallback,context);
// initialize E3Kit with the EThreeParams
EThree ethree = new EThree(params);
```

User Login

1.Call Tencent Cloud Chat Sdk login method for account login



```
String userID = "your user id";
//Generating UserSig | Tencent Cloud
String userSig = "userSig from your server";
V2TIMManager.getInstance().login(userID, userSig, new V2TIMCallback() {
   @Override
   public void onSuccess() {
       Log.i("imsdk", "success");
   }
    @Override
```

```
    public void onError(int code, String desc) {
        // The following error codes indicate an expired `userSig`, and you need to
        // 1. ERR_USER_SIG_EXPIRED (6206)
        // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED (70001)
        // Note: Do not call the login API in case of other error codes; otherwise,
        Log.i("imsdk", "failure, code:" + code + ", desc:" + desc);
    }
});
```

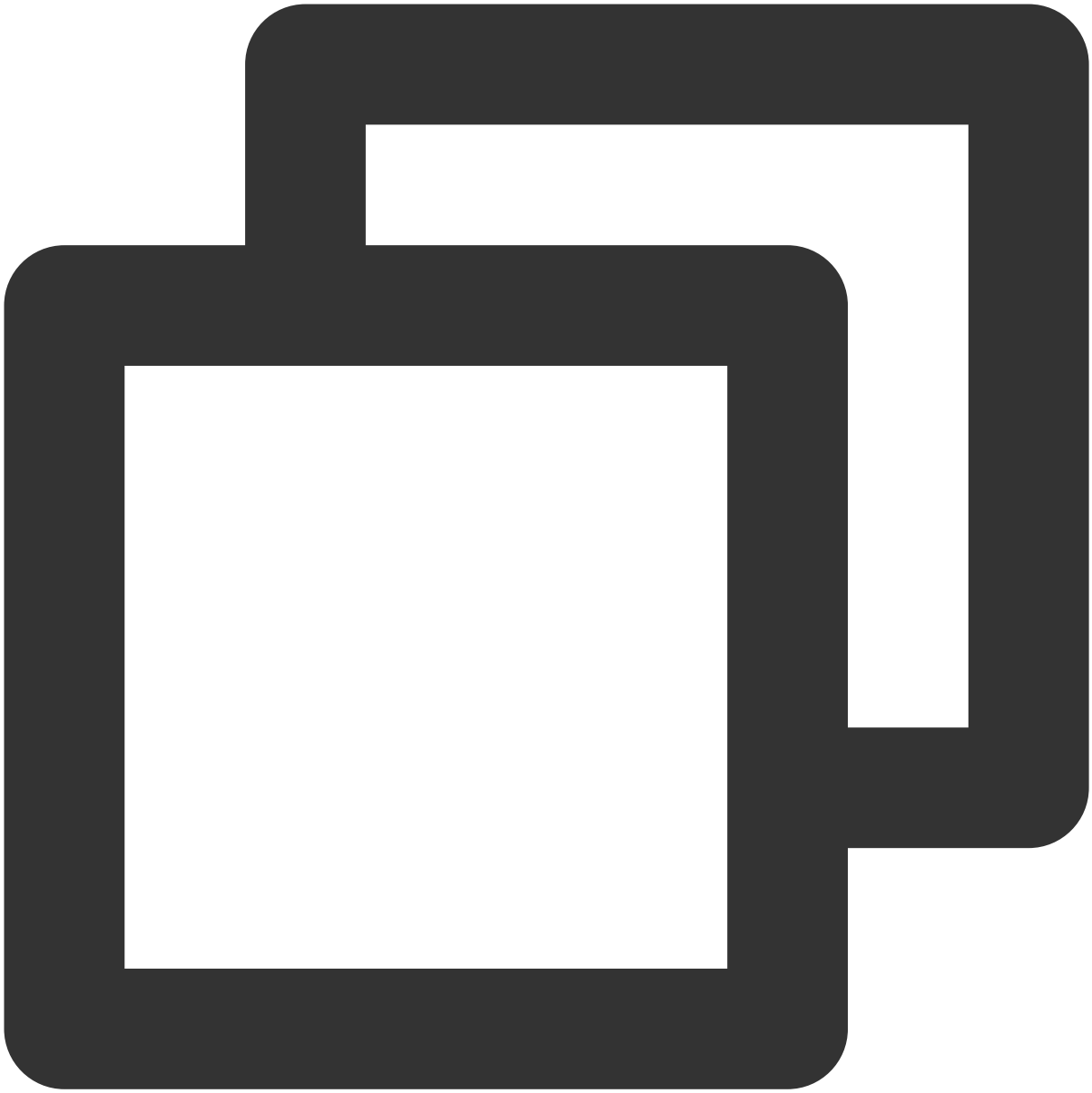
2.Call the eThree.register method to register the user to virgilsecurity. The corresponding operation document link: UserAuthentication-E3Kit | VirgilSecurity

Note：User of im_ ID and user registered on virtilsecurity_ The id should be consistent

Start a one-to-one chat

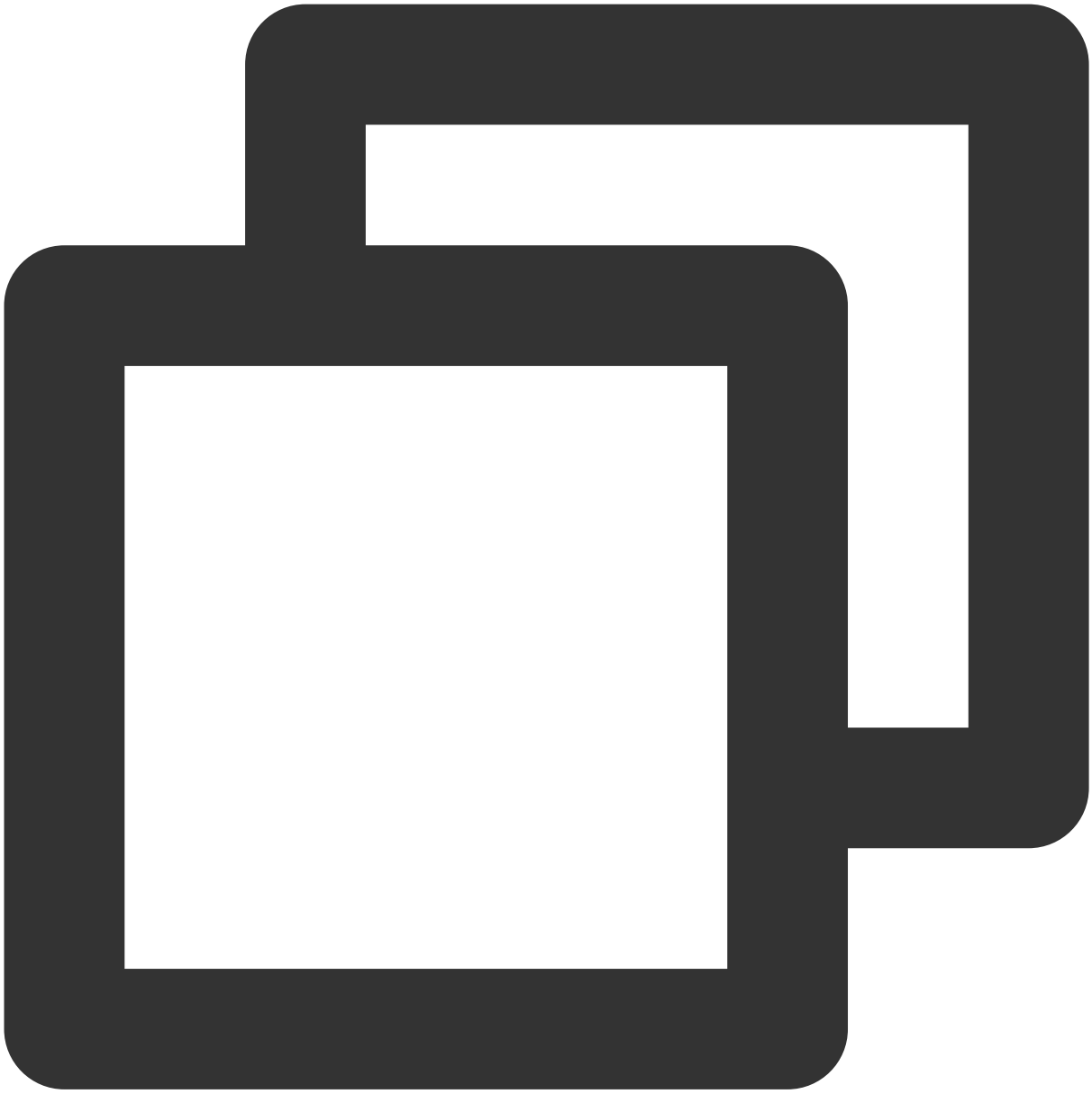
1.Call the ethree.createRatchetChannel method in the E3Kit to create a one to one session (User1 and User2), and the corresponding operation document link: https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/double-ratchet/?#create -channel

User1 creates a channel with User2



```
// create one-to-one channel
ethree.createRatchetChannel(users.get("User2"))
      .addCallback(new OnResultListener<RatchetChannel>() {
          @Override public void onSuccess(RatchetChannel ratchetChannel) {
              // Channel created and saved locally!
          }

          @Override public void onError(@NotNull Throwable throwable) {
              // Error handling
          }
      });
```

User2 can join the channel

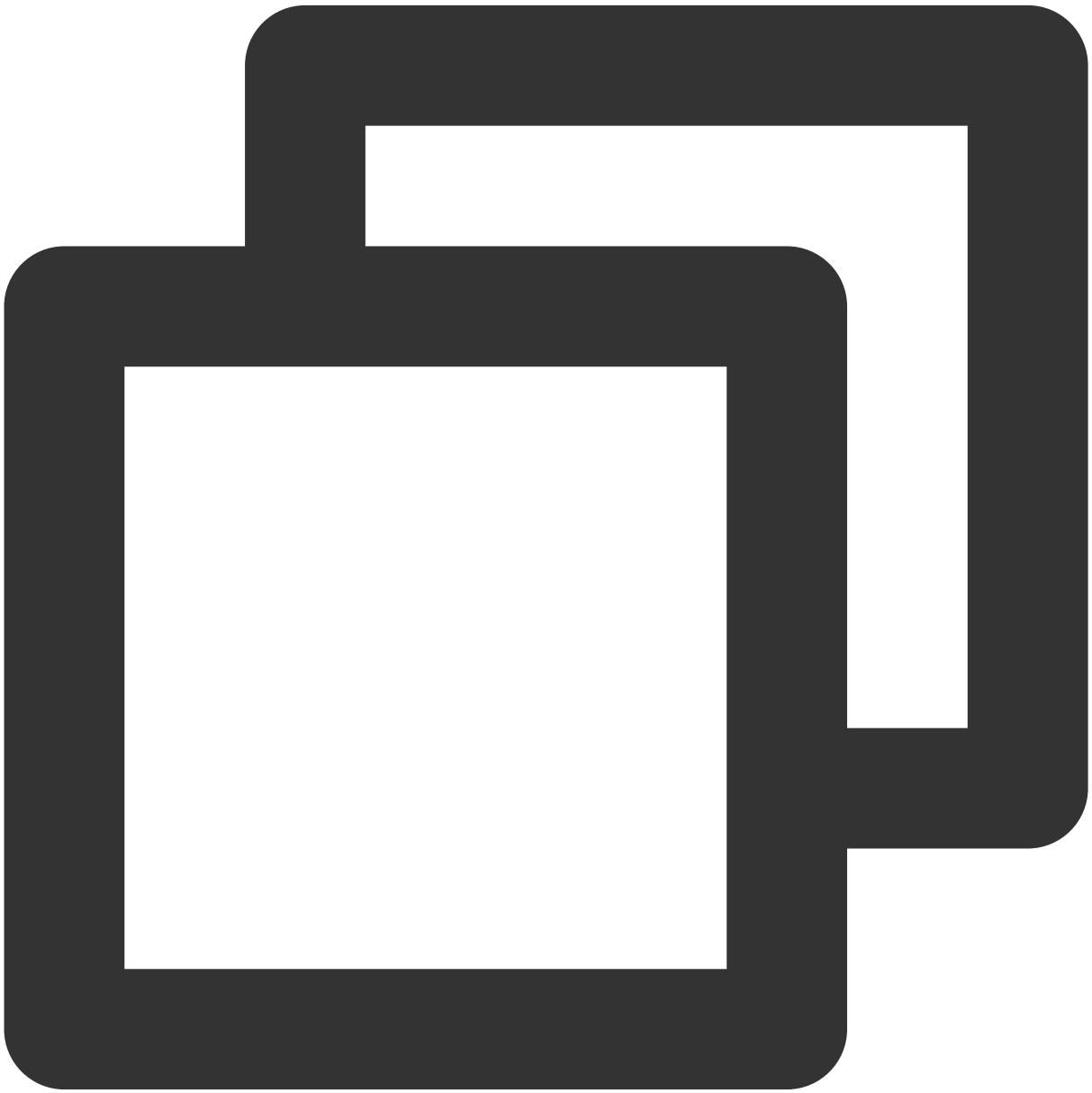


```
// join channel
ethree.joinRatchetChannel(users.get("User1"))
      .addCallback(new OnResultListener<RatchetChannel>() {
    @Override public void onSuccess(RatchetChannel ratchetChannel) {
        // Channel joined and saved locally!
    }

    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
```

```
    }
});
```

One-to-one chat message encryption and decryption

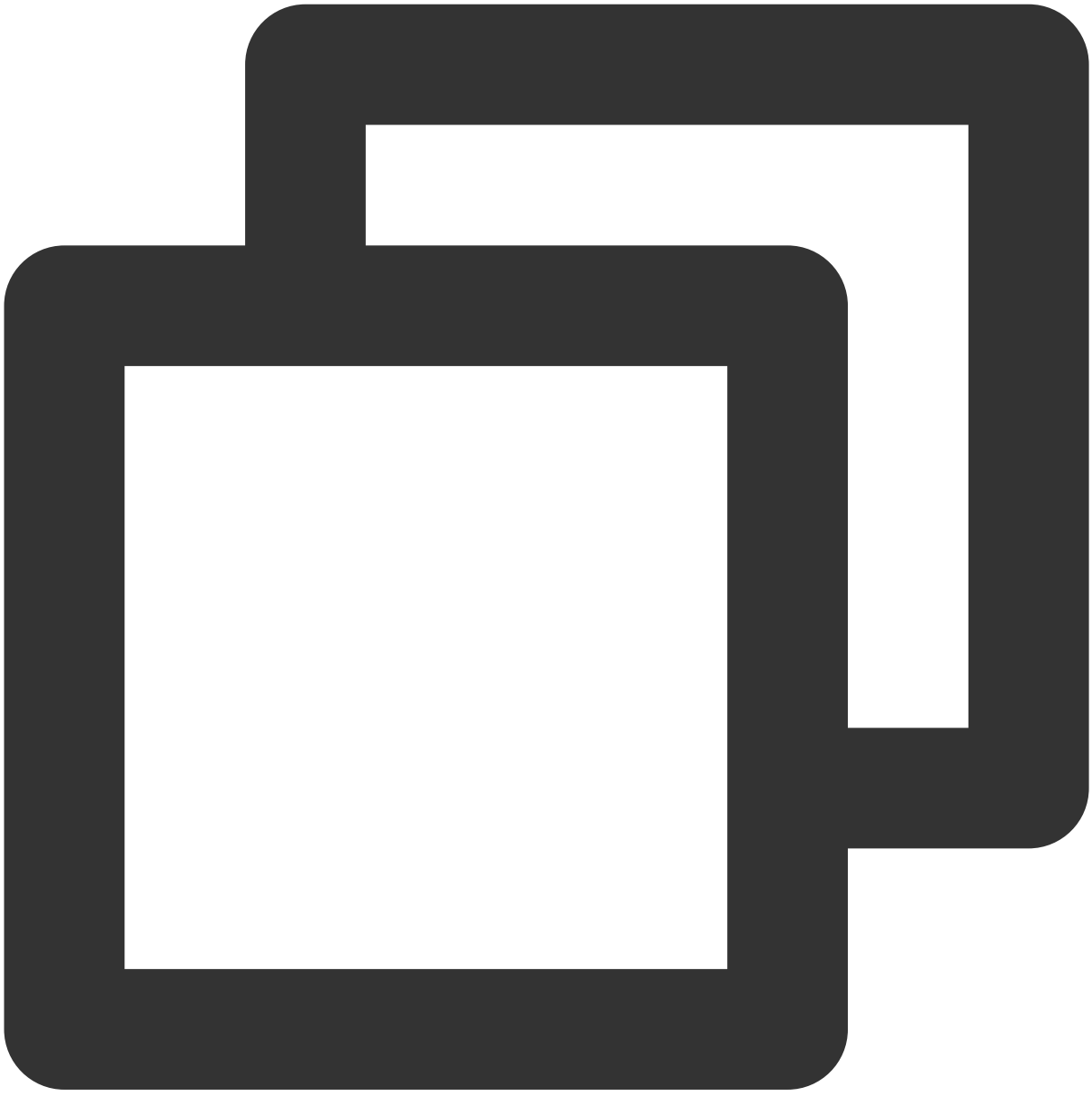
1.Use the channel created above to encrypt messages and link documents：
https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/double-ratchet/?#encrypt-and-decrypt-messages



```
// one-to-one chat message encryption
// prepare a message
```

```
String messageToEncrypt = "Hello, User2!";

String encrypted = channel.encrypt(messageToEncrypt);
```

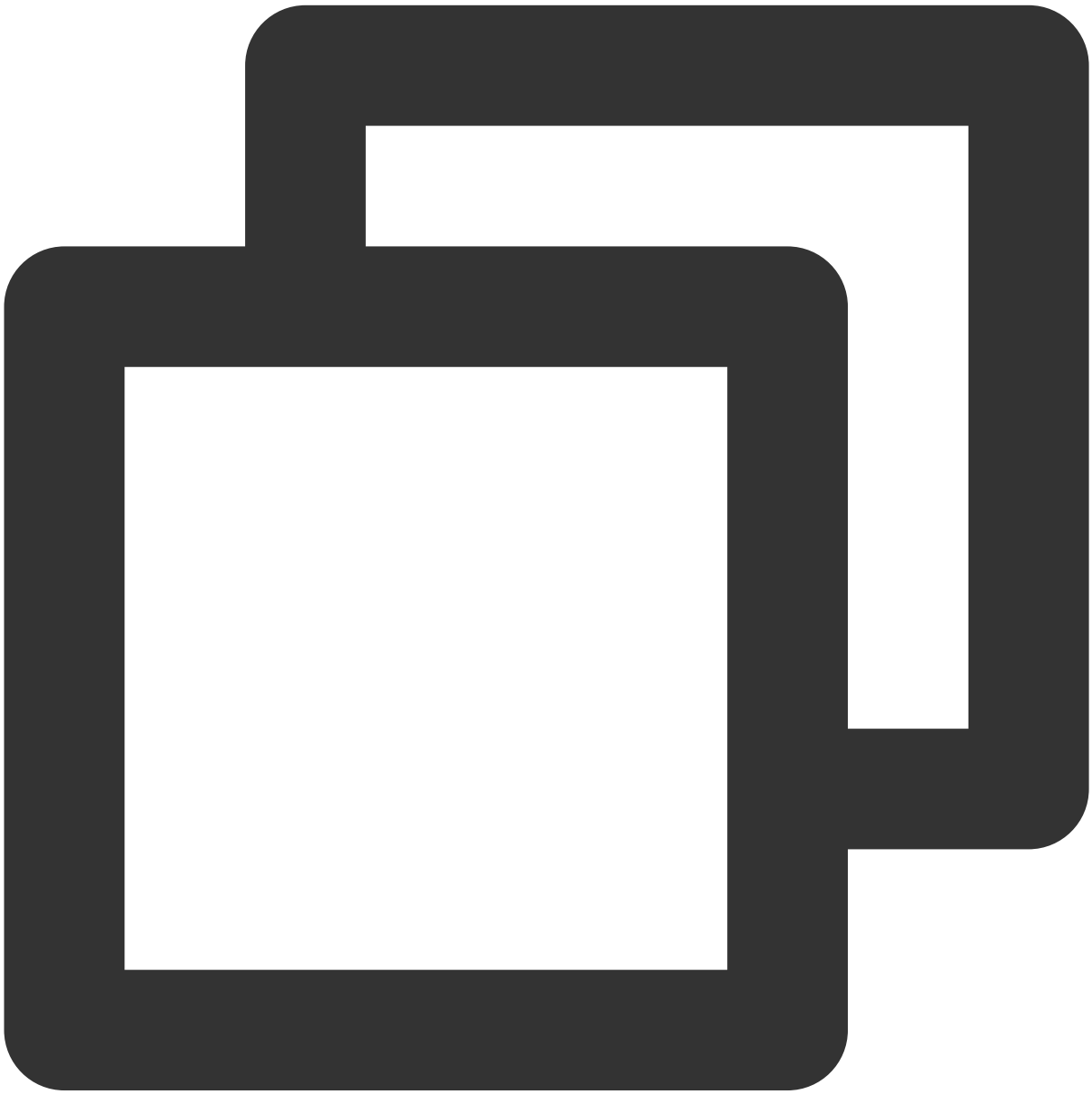
2.The encrypted message content is sent to Tencent Cloud Chat Sdk and sent with a customized message



```
// `msgID` returned by the API for on-demand use
String msgID = V2TIMManager.getInstance().sendC2CCustomMessage("virgil encrypted ms
@Override
public void onSuccess(V2TIMMessage message) {
    // The one-to-one text message sent successfully
}
```

```
    @Override
public void onError(int code, String desc) {
    // Failed to send the one-to-one text message
}
});
```

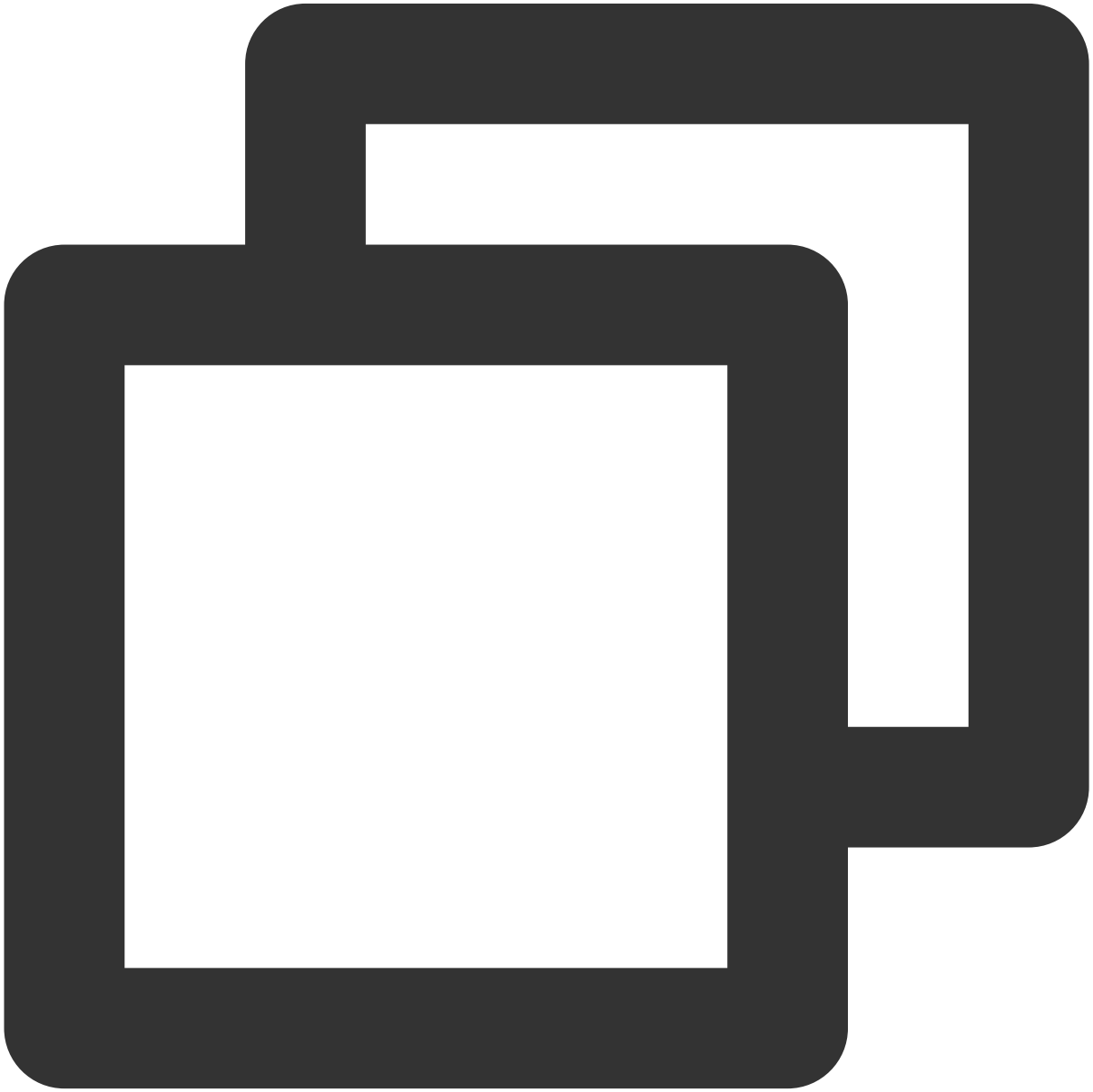
3.After the peer receiving the customized message



```
// Set the event listener
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

/**
```

```
* Receive the custom one-to-one message
* @param msgID Message ID
* @param sender Sender information
* @param customData The sent content
*/
public void onRecvC2CCustomMessage(String msgID, V2TIMUserInfo sender, byte[] custo
Log.i("onRecvC2CCustomMessage", "msgID:" + msgID + ", from:" + sender.getNickName()
//call E3Kit to decrypt msg
}
```

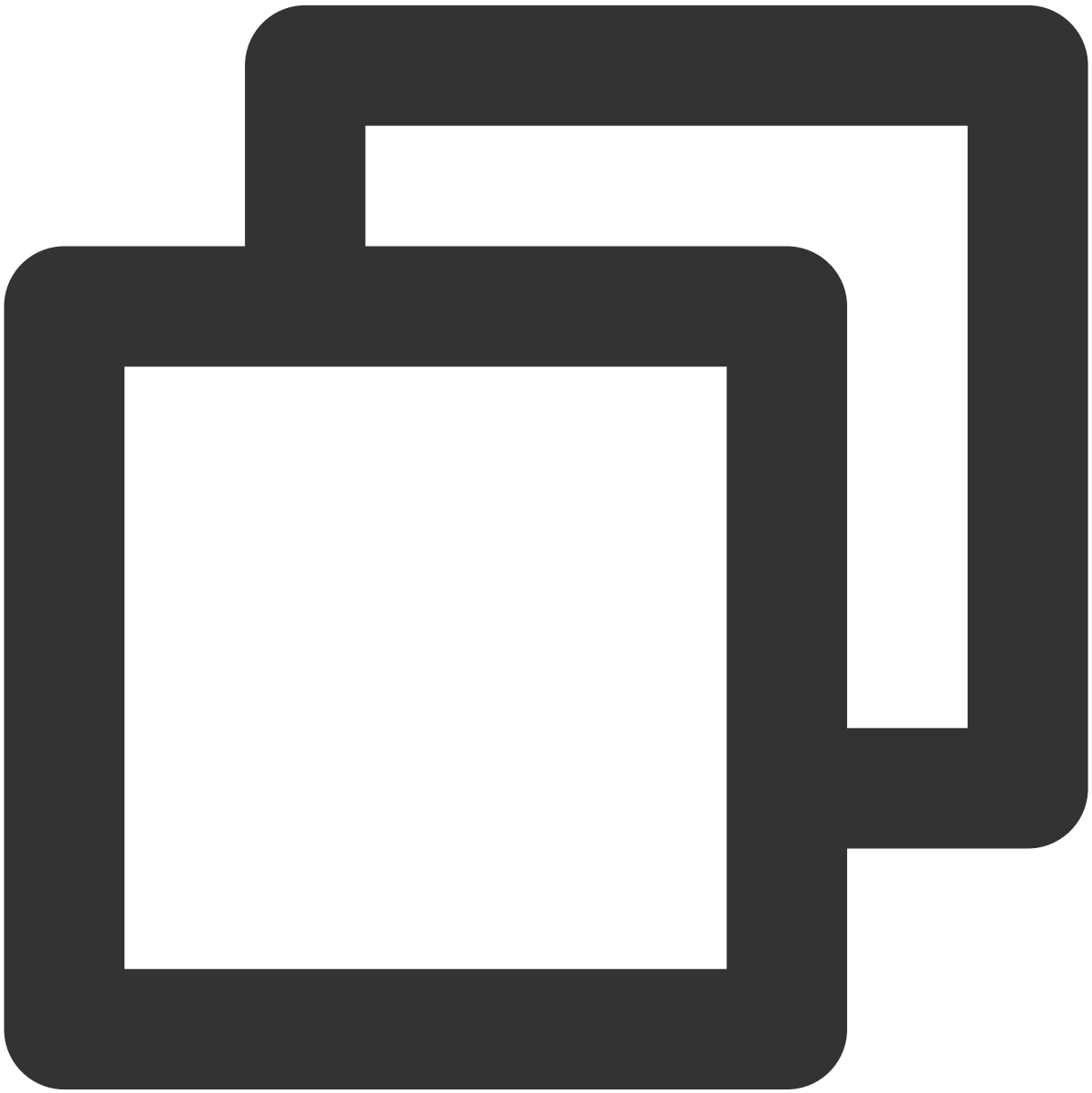
4.the peer calls E3Kit to decrypt and render it, as follows:



```
// Decrypt message
String decrypted = channel.decrypt(encrypted);
```

Start a channel(group)

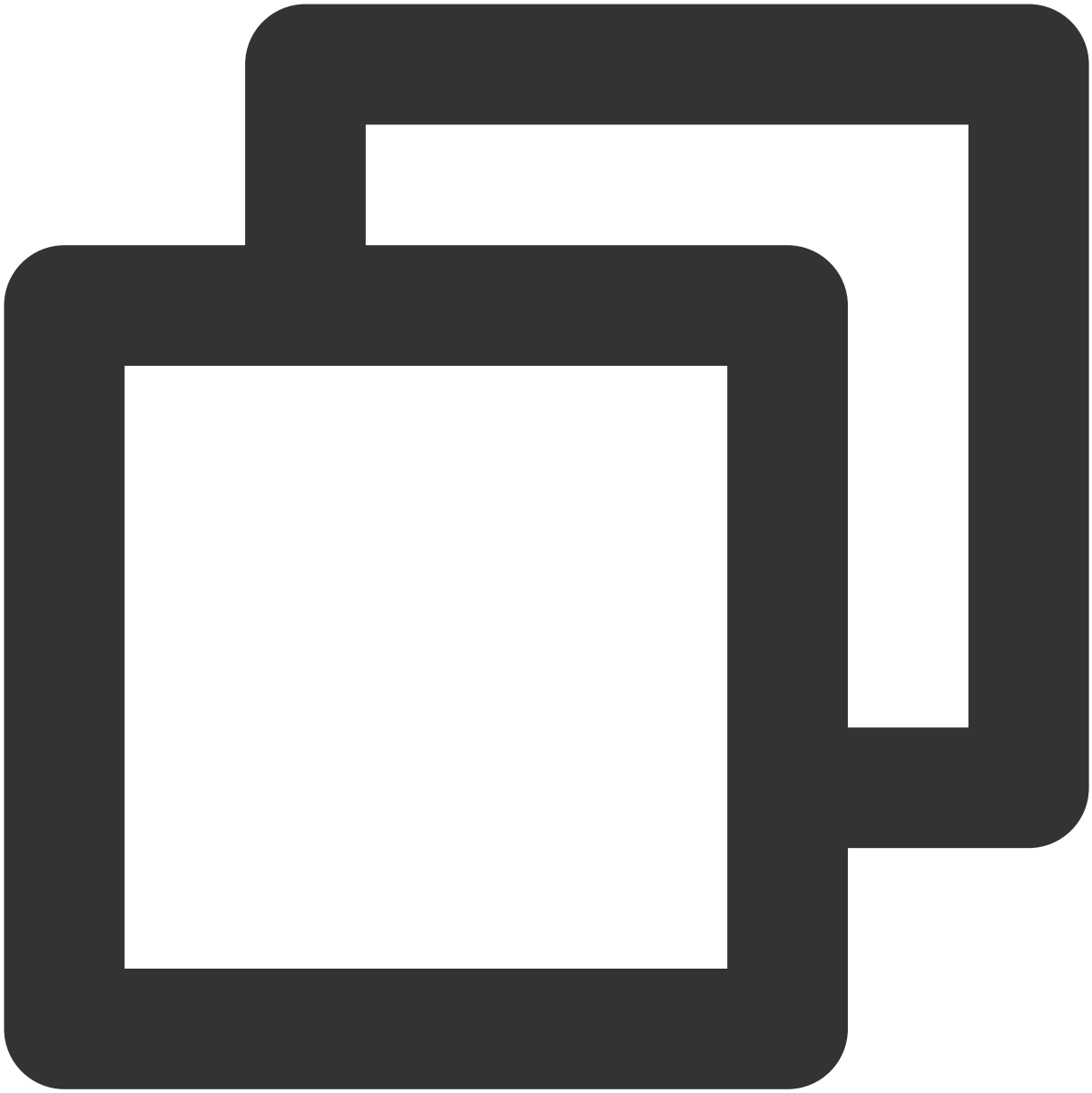
1.Call the ethree.createGroup to create group， document link：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#create-group-chat



```
// create group
ethree.createGroup(groupId, users).addCallback(new OnResultListener<Group>() {
```

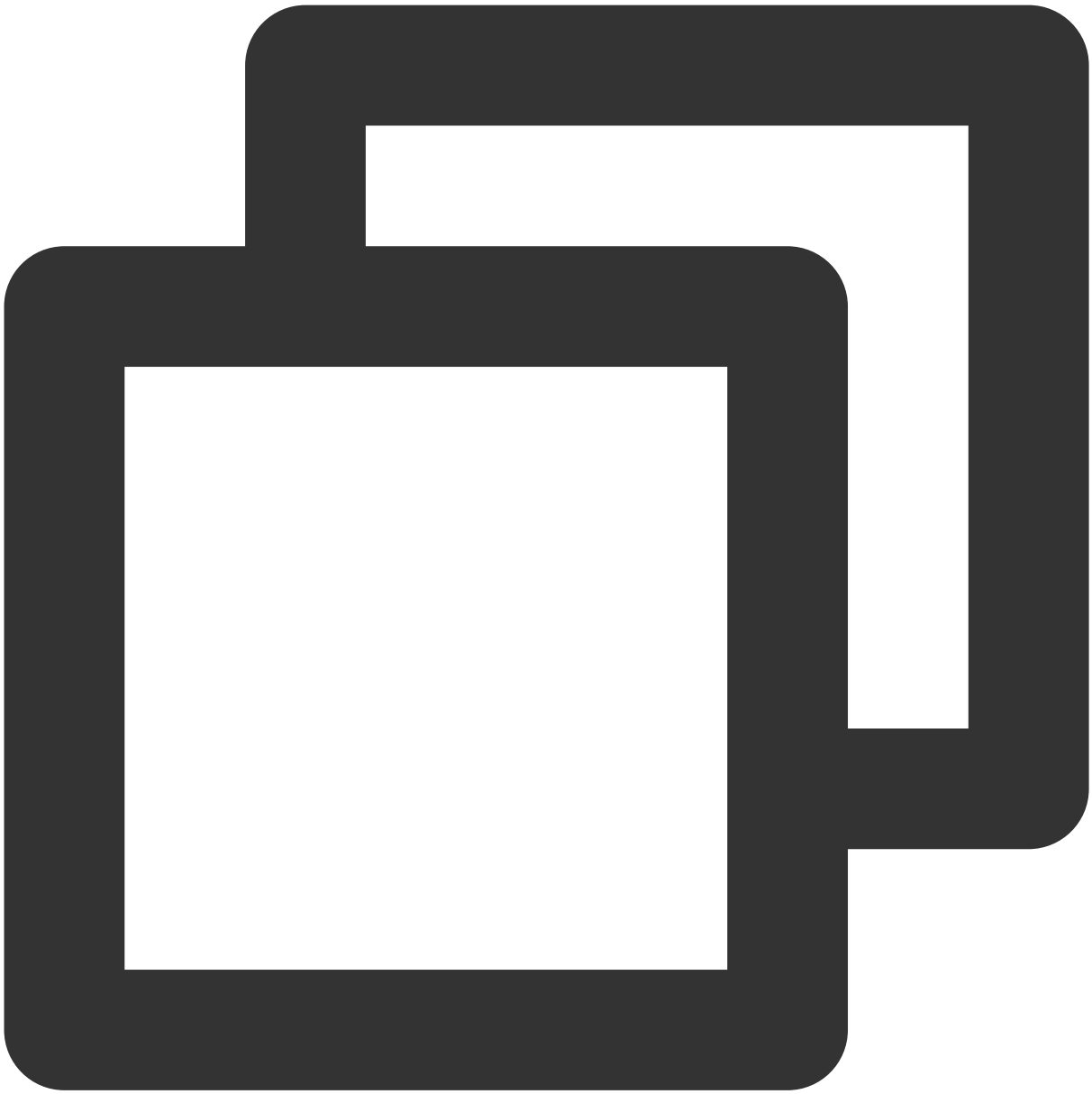
```
    @Override public void onSuccess(Group group) {
        // Group created and saved locally!
    }

    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
    }
});
```

2.If you need to add group members, the code below is as follows. You can call the remove method to delete a group member. Document link：https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#add-new-participant



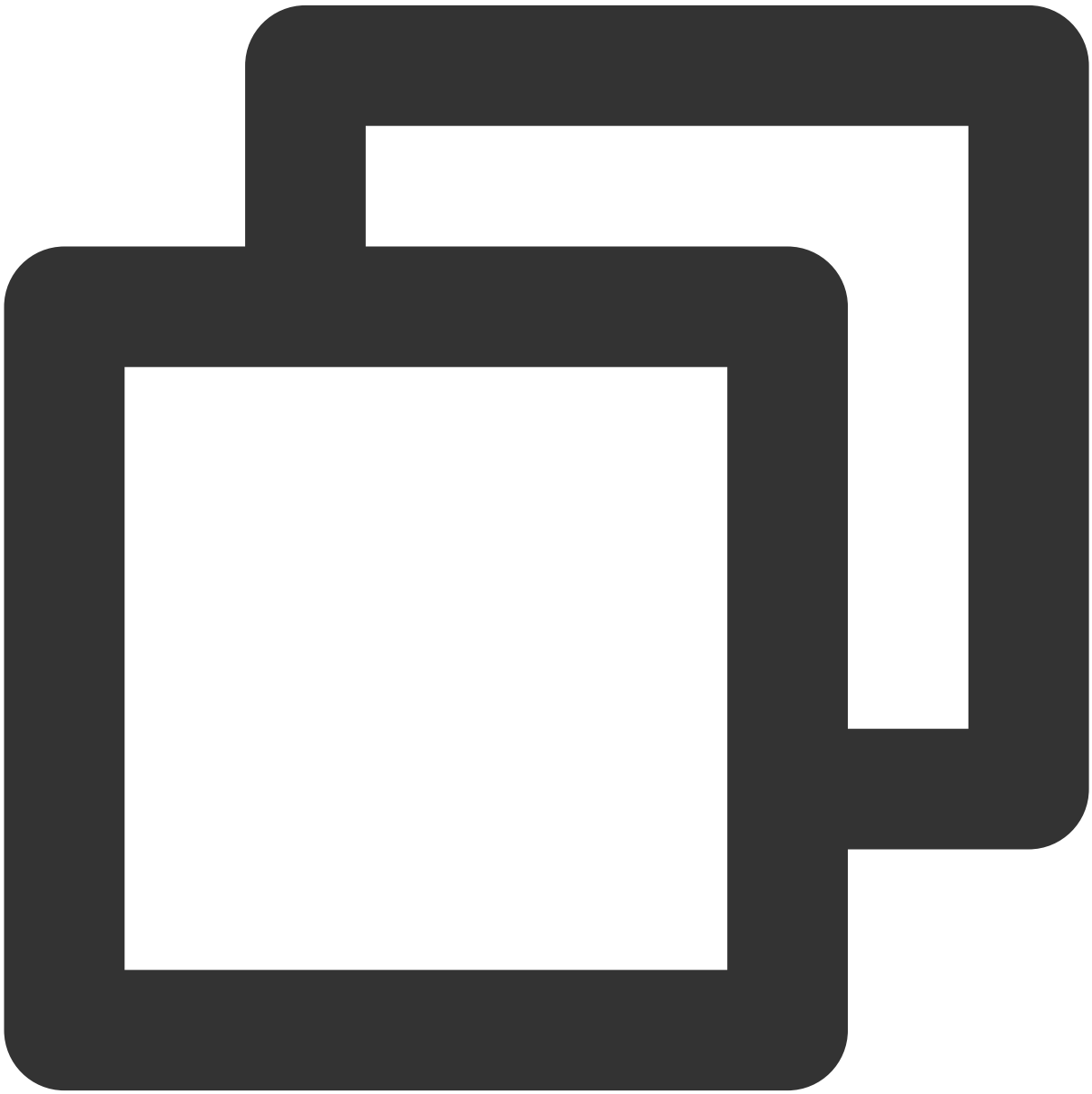
```
// add group member
group.add(users.get("Den")).addCallback(new OnCompleteListener() {
    @Override public void onSuccess() {
        // Den was added!
    }

    @Override public void onError(@NotNull Throwable throwable) {
        // Error handling
    }
});
```

3.Call createGroup of Tencent Cloud Chat Sdk to create a group, joinGroup or inviteUserToGroup to add group members



```
V2TIMManager.getInstance().createGroup(V2TIMManager.GROUP_TYPE_WORK, null, "groupA"
 @Override
 public void onSuccess(String s) {
     // Group created successfully
 }
  @Override
 public void onError(int code, String desc) {
     // Failed to create the group
```

```
 }
});
// Listen for the group creation notification
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
 @Override
 public void onGroupCreated(String groupID) {
     // A group was created. `groupID` is the ID of the created group.
 }
});
```



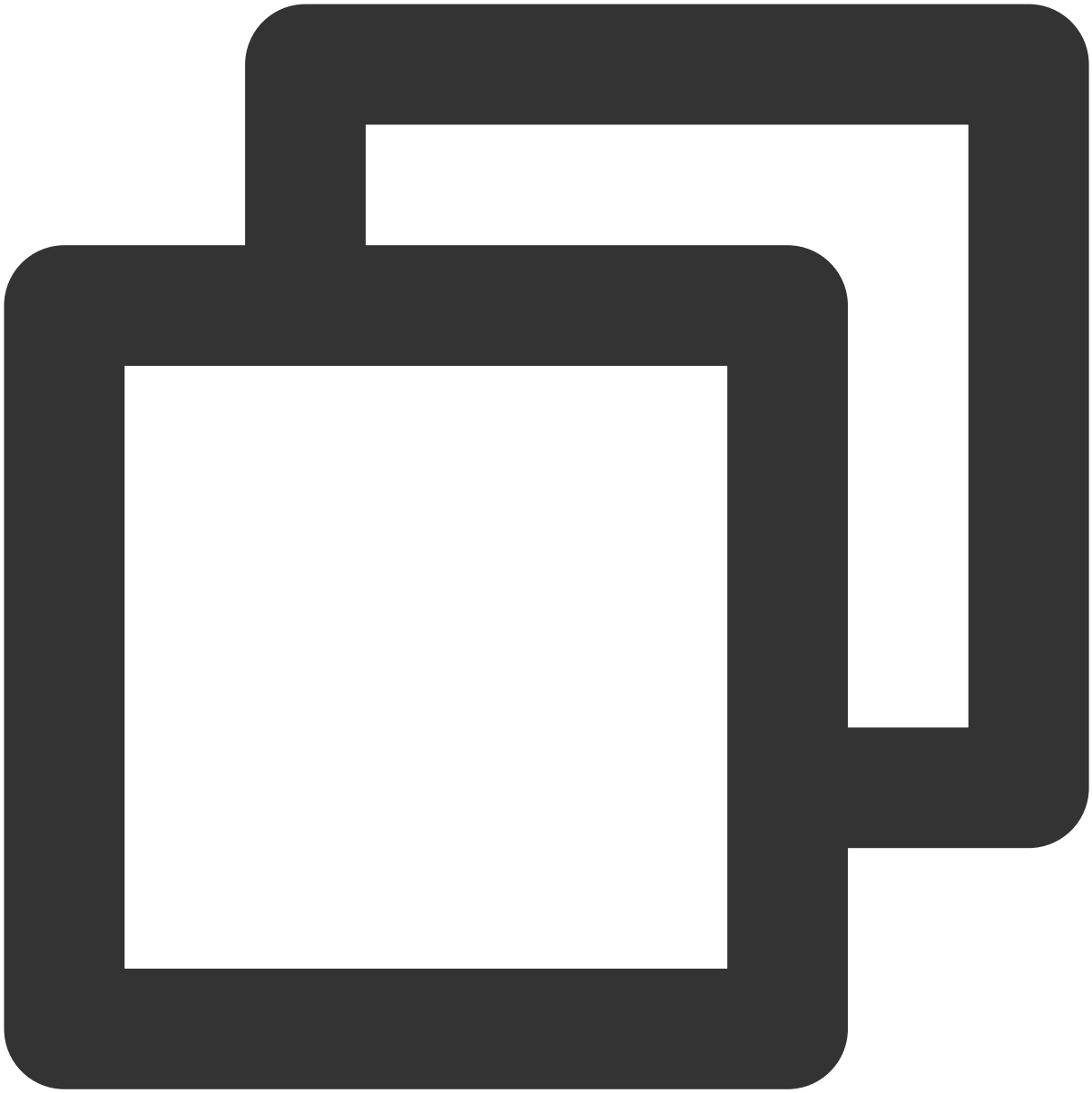
```
// Invite the `userA` user to join the `groupA` group
```

```
List<String> userIDList = new ArrayList<>();
userIDList.add("userA");
V2TIMManager.getGroupManager().inviteUserToGroup("groupA", userIDList, new V2TIMVal
 @Override
 public void onSuccess(List<V2TIMGroupMemberOperationResult> v2TIMGroupMemberOperat
     // Invited the user to the group successfully
 }
  @Override
 public void onError(int code, String desc) {
     // Failed to invite the user to the group
 }
});
// Listen for the group invitation event
V2TIMManager.getInstance().addGroupListener(new V2TIMGroupListener() {
 @Override
 public void onMemberInvited(String groupID, V2TIMGroupMemberInfo opUser, List<V2TI
     // A user was invited to the group. This callback can contain some UI tips.
 }
});
```

Group chat message encryption and decryption

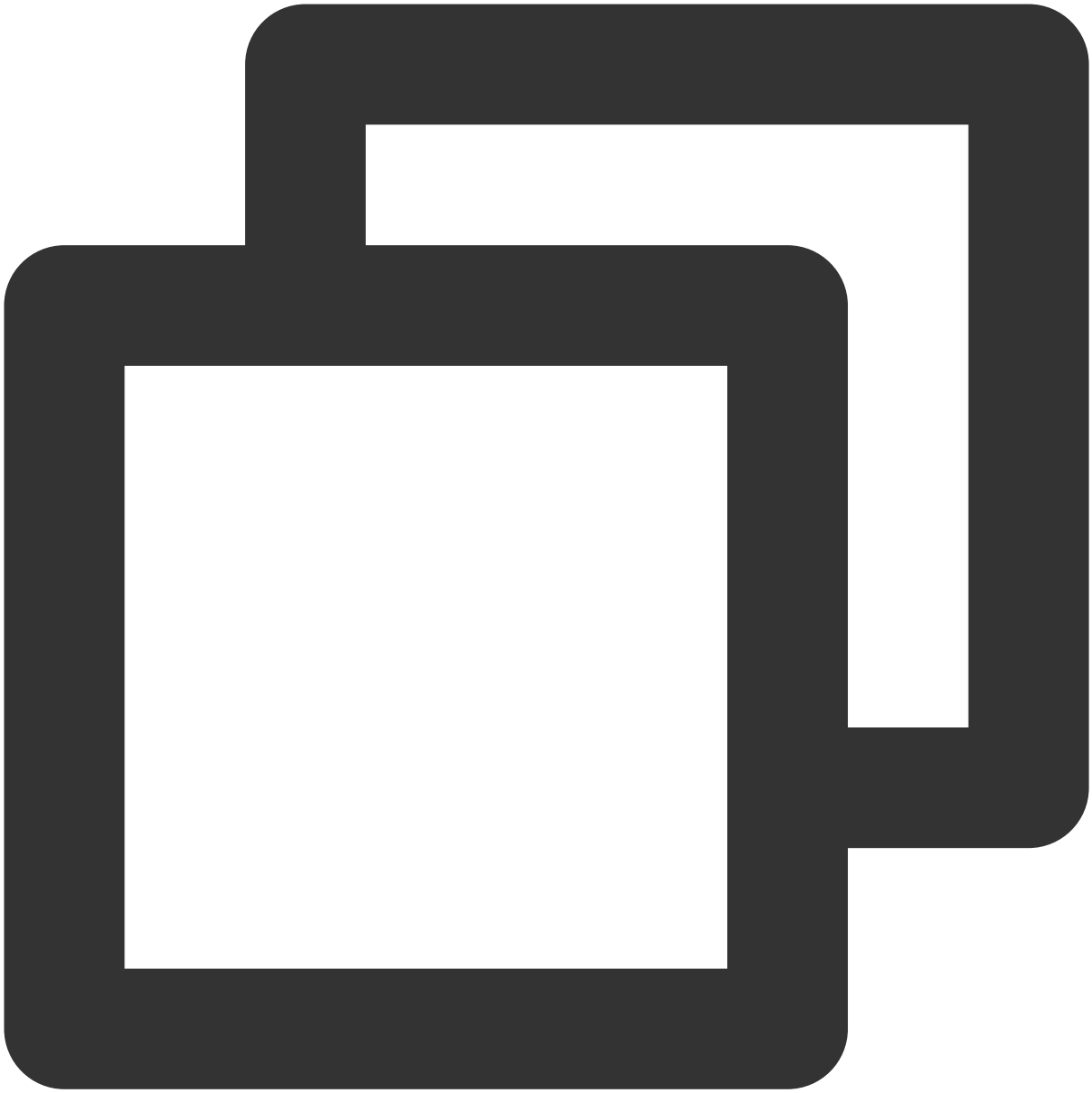
1.Use the group created above to encrypt messages and link documents：

https://developer.virgilsecurity.com/docs/e3kit/end-to-end-encryption/group-chat/?#encrypt-and-decrypt-messages



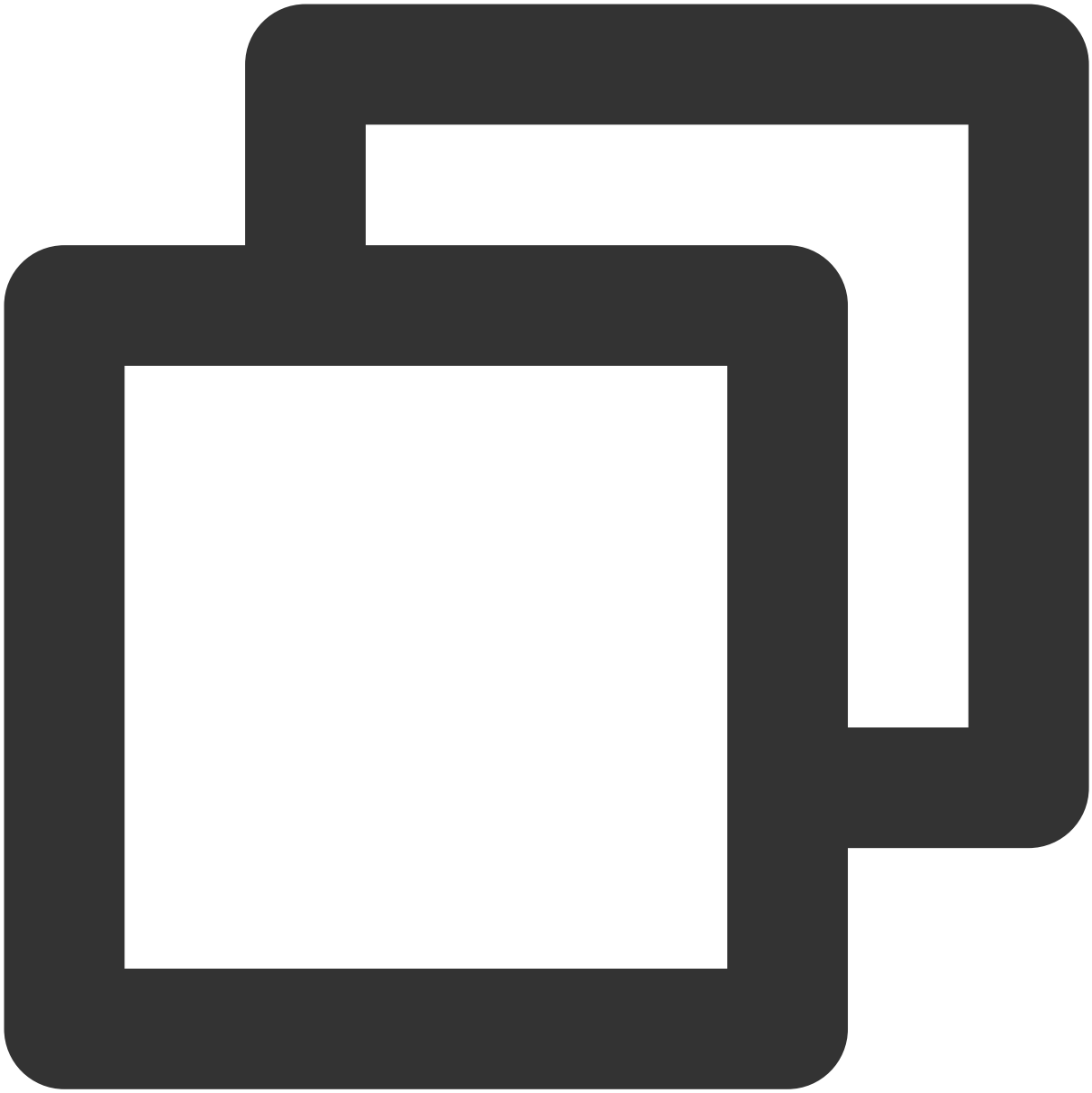
```
//Group message encryption
// prepare a message
String messageToEncrypt = "Hello, Bob and Carol!";

String encrypted = group.encrypt(messageToEncrypt);
```

2.The encrypted message content is sent to tencent Chat Sdk and sent with a customized message



```
String msgID = V2TIMManager.getInstance().sendGroupCustomMessage("virgil encrypted
@Override
public void onSuccess(V2TIMMessage message) {
    // The custom group message sent successfully
}
```

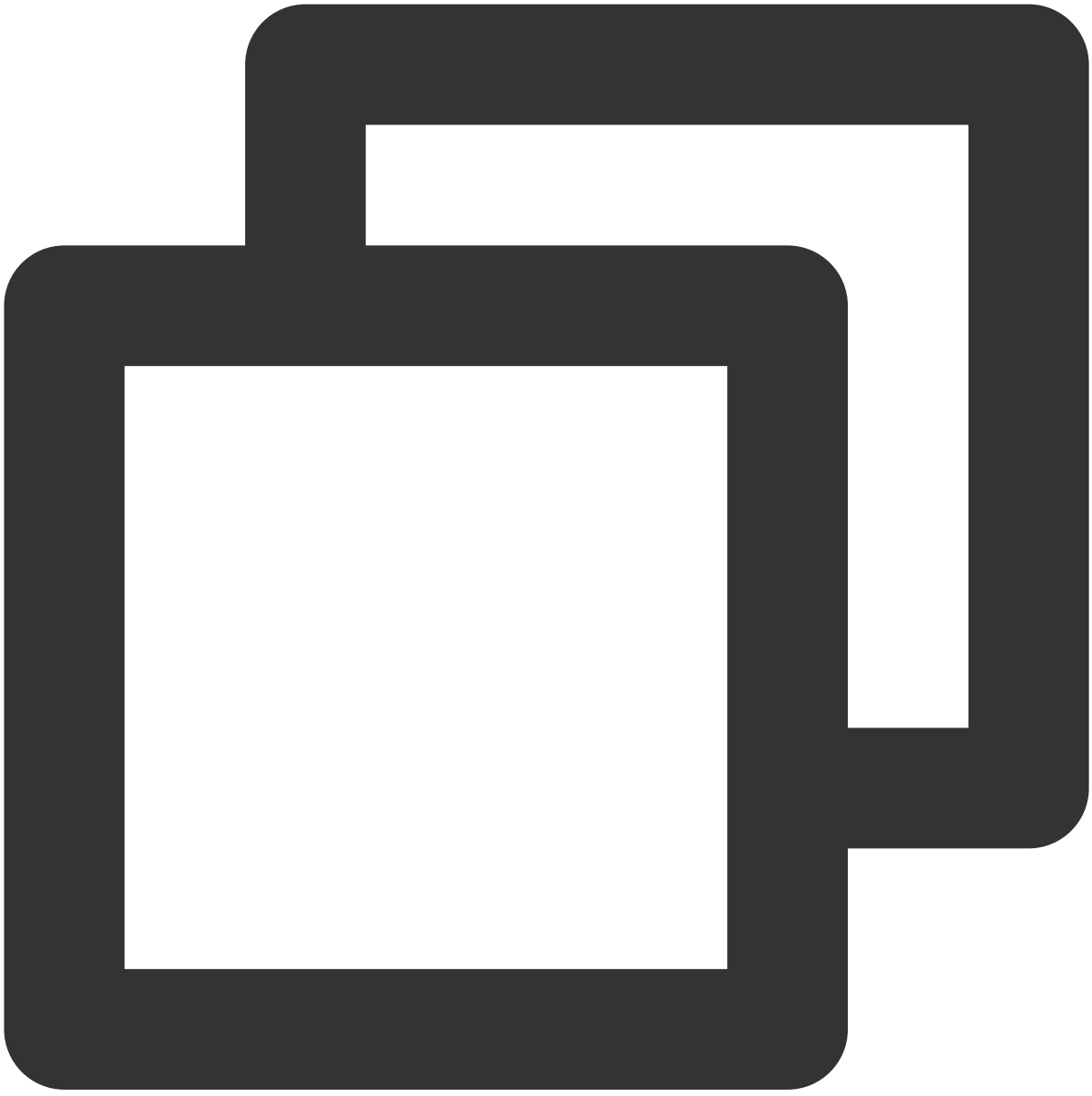
3.After the peer tencent cloud Chat Sdk receiving the customized message,



```
// Set the event listener
V2TIMManager.getInstance().addSimpleMsgListener(simpleMsgListener);

/**
* Receive the custom group message
* @param msgID Message ID
* @param groupID Group ID
* @param sender The group member information of the sender
* @param customData The sent content
*/
public void onRecvGroupCustomMessage(String msgID, String groupID, V2TIMGroupMember
```

```
Log.i("onRecvGroupCustomMessage", "msgID:" + msgID + ", groupID:" + groupID + ", fr
//call E3Kit to decrypt msg
}
```

4.The peer decrypt and render it, as follows：

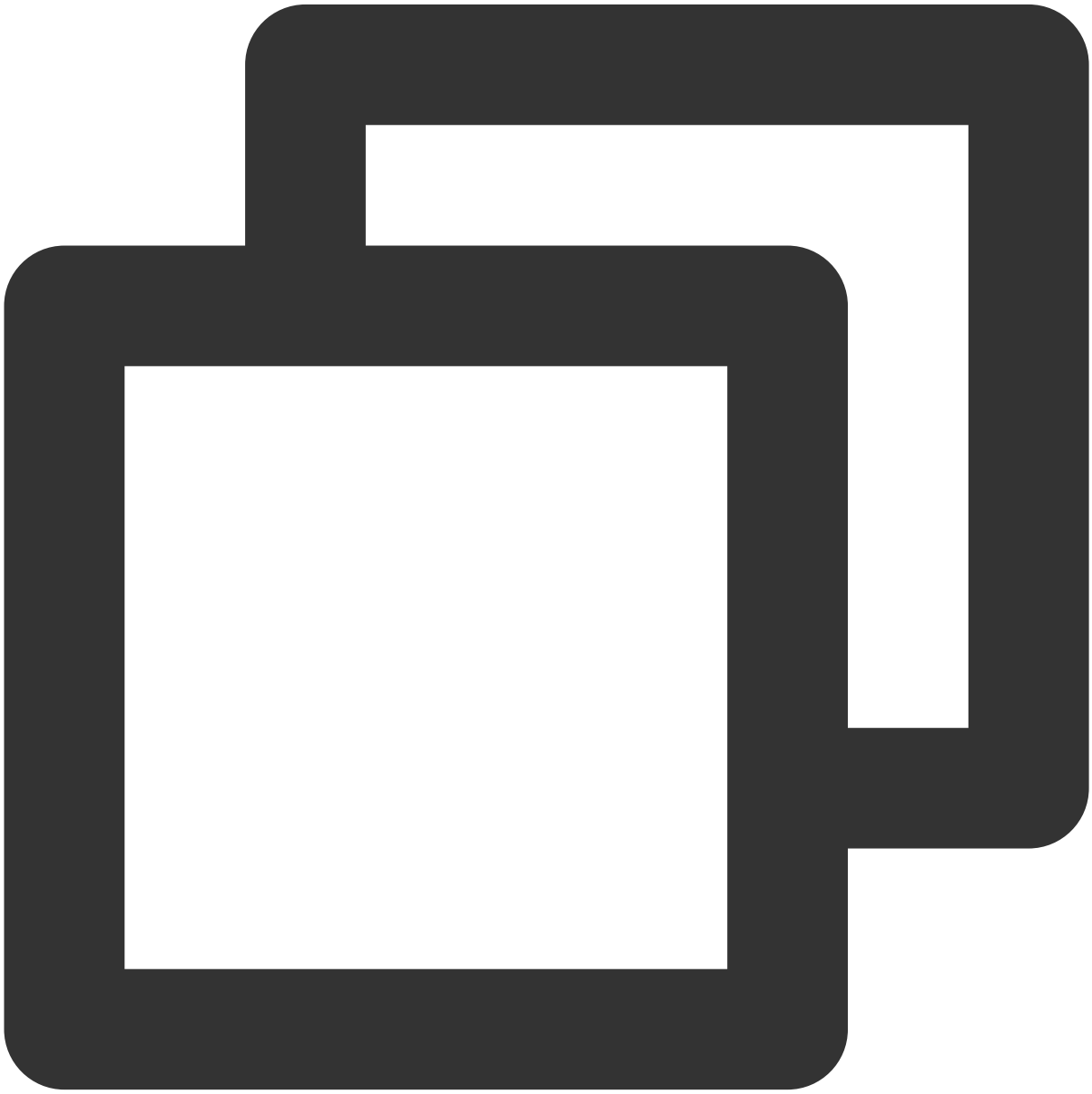


```
//Group message decryption
String decrypted = group.decrypt(encrypted, users.get("Alice"));
```

Note: After using this end-to-end encryption scheme, the local chat record search function of imsdk will not be available

IM - Developer Groups

Join a Tencent Cloud IM developer group for:



```
- Reliable technical support
- Product details
- Constant exchange of ideas
```

Telegram group (EN): join

WhatsApp group (EN): join

Telegram group (ZH): join

WhatsApp group (ZH): join

대규모 엔터테인먼트 협업 커뮤니티

최종 업데이트 날짜: : 2024-02-22 12:02:25

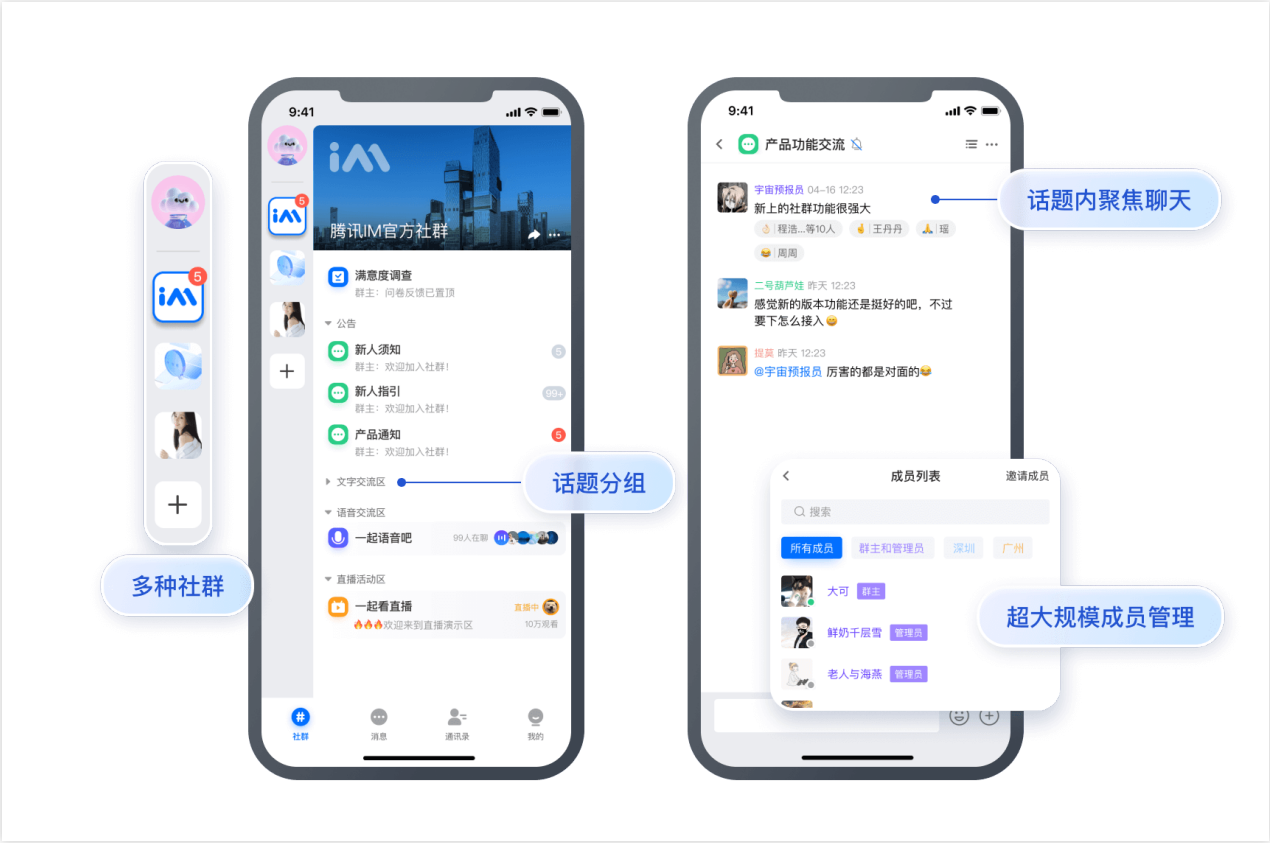
기능 소개

커뮤니티 모드(엔터테인먼트 협업을 위한 새로운 도구)는 **커뮤니티-그룹-토픽** 3단계 구조를 지원하고, 메시지를 서로 분리할 수 있으며, 초대형 규모의 참석자를 운영하고, 친구 관계를 공유하며, 또한 참석자를 그룹화하고, 참석자 그룹 보기, 말하기 및 관리 등 권한을 설정할 수 있습니다.

적용 시나리오

친구 사귀기: 새로운 사용자 증가 모델

초대형 동호회 모임을 지원하며 **커뮤니티-그룹-토픽**을 통해 관심사를 수직적으로 세분화 할 수 있습니다.
대규모 공개 커뮤니티에서는 사용자에게 폐쇄적인 작은 토픽을 제공하고, 이러한 편안한 중간 지대는 사용자가 원하는 토픽을 선택해 커뮤니케이션할 수 있어, 참석자의 적극적인 참여를 유도합니다.



게임 기반 소셜 네트워크: 더 높은 사용자 충성도 및 참여도

다양한 커뮤니티 토픽을 통해 유저는 뉴스에 액세스하고, 팀원을 찾고, 플롯에 대해 토론하고, 팁을 공유할 수 있습니다. 게임을 시작하기 전에 사용자는 조언을 검색할 수 있습니다. 게임 중에 항상 다른 사람과 대화할 수 있으며(채팅방은 항상 온라인 상태), 게임이 끝난 후에도 토픽에 대한 깊이 있는 커뮤니케이션을 계속할 수 있습니다.

팬 운영: 효율적인 운영 툴 보유

각각의 독립된 그룹('선전 사용자 1그룹' '선전 사용자 2그룹' '광저우 사용자 1그룹' '상하이 사용자 1그룹' 등 여러 그룹 대체)을 운영할 필요 없이 하나의 커뮤니티에서 다양한 토픽을 다룰 수 있기 때문에 더 이상 여러 그룹을 동시에 관리해야 하는 걱정을 할 필요가 없어 팬 운영이 더 쉽고 정확해집니다!

조직 관리: 명확한 레이어 커뮤니케이션 구현

조직의 모든 구성원은 동일한 커뮤니티에 가입할 수 있습니다. 커뮤니티 계층 및 권한 설정을 통해 계층적 커뮤니케이션을 구현합니다.

기술 경쟁력

초대형 규모의 커뮤니티 구성원

Tencent IM 커뮤니티는 용량이 1만 배 가까이 확장되어, 취미 친구 사귀기, 팬 마케팅, 게임 기반 소셜 네트워킹 및 조직 관리와 같은 사용 사례에서 대규모 회원을 수용해야 하는 요구 사항을 완벽하게 충족합니다.

메시지 신뢰성

Tencent IM 커뮤니티는 구성원 용량을 대폭 확대하면서 Tencent의 강력한 메시징 능력을 이어받아 20여 년의 기술 축적을 바탕으로 구축한 완벽하고 신뢰할 수 있는 메시징 시스템입니다. 99.99% 이상의 메시지 송수신 성공률 및 서비스 신뢰성을 고객에게 제공하여 고객이 수억 대의 대규모 동시성에 쉽게 대처할 수 있도록 지원합니다.

메시지 푸시 성능

Tencent IM 커뮤니티는 '빠르고 느린 채널' + '2단계 결합 푸시'의 새로운 메시지 푸시 아키텍처를 채택하여 시간과 공간의 균형을 효과적으로 유지하고, 시스템 푸시 성능을 향상시키며, 단말 성능 소모를 줄이고, 초대형 그룹의 사용자에게도 일반 그룹과 동일한 메시지 인터랙션 경험을 제공할 수 있습니다.

메시지 상태 및 사용자 권한 관리

Tencent IM 커뮤니티는 메시지 편집, 회수, 포워딩 등 풍부한 확장 기능을 제공합니다. 음소거, 메시지 방해 금지, 읽지 않음 메시지 수, 프로필 편집 등은 사용자가 전역, 커뮤니티, 토픽 수준을 각각 사용자 정의할 수 있도록 지원합니다.

네이티브 SDK 통합 가이드

주의 :
커뮤니티(Community) 토픽(Topic) 기능은 v6.2.2363 이상의 IM 네이티브 SDK 인핸스드 버전 이상에서만 지원됩니다. 이를 사용하려면 플래그십 버전 구매 및 구성 변경 요청 티켓의 지침에 따라 활성화를 신청하십시오.
다음은 Android를 예시로 커뮤니티 토픽의 인터페이스 기능을 소개합니다.

Demo를 다운로드하여 커뮤니티 기능을 빠르게 체험

Android 휴대폰으로 아래 QR코드를 스캔하여 Demo를 다운로드 후 바로 체험할 수 있습니다.
다음은 API 호출의 관점에서 커뮤니티 토픽의 사용을 소개합니다.

커뮤니티 토픽의 API 사용

1. 먼저 createGroup 인터페이스를 호출하여 토픽을 지원하는 커뮤니티를 생성하고, 커뮤니티 관리 '커뮤니티 생성' 단계를 참고하여 구현할 수 있습니다.
2. 그 다음 getJoinedCommunityList 인터페이스를 호출하여 이러한 유형의 생성 및 가입 커뮤니티 목록을 가져올 수 있습니다.

주의 :

커뮤니티는 그룹 구성원을 관리하는 데 사용되지만 커뮤니티에서 메시지를 주고받을 수는 없습니다. 커뮤니티의 다른 기능에 대해서는 '기타 관리 인터페이스' 목록을 참고하십시오.
3. 커뮤니티가 성공적으로 생성되면 커뮤니티에서 createTopicInfoCommunity를 호출하여 여러 개의 토픽을 생성할 수 있습니다. Topic Management의 'Creating a topic' 단계를 참고하십시오.
4. 토픽 관리에는 '토픽 삭제', '토픽 정보 수정', '토픽 목록 가져오기' 및 '토픽 콜백 수신' 기능도 포함됩니다. 동시에 토픽은 사용자가 메시지를 주고받을 수 있는 커뮤니케이션의 장이며, 관련 인터페이스는 토픽 메시지소개를 참고하십시오.
5. 그룹 커뮤니티 구성원: 그룹 구성원 사용자 정의 필드에 그룹 정보를 지정하여 그룹의 커뮤니티 구성원을 표시할 수 있습니다. 이 효과를 구현하려면 모든 커뮤니티 구성원을 로컬로 끌어와 그룹별로 정렬해야 합니다. 그룹 구성원 수가 많은 경우 서버측에서 구성원 그룹화를 구현하는 것이 좋습니다.

Web SDK 통합 가이드

주의 :

커뮤니티(Community) 토픽(Topic) 기능은 v2.19.1 이상의 IM Web SDK에서만 지원됩니다. 사용하려면 플래그십 에디션을 구매하고 **콘솔>그룹 기능 구성>커뮤니티** 기능을 활성화하십시오.
커뮤니티 토픽의 인터페이스 기능은 다음과 같습니다.

빠른 체험을 위한 Demo 소스 코드 다운로드 및 설정

시작하기를 참고하여 IM 기능을 빠르게 경험할 수 있습니다.
다음은 API 호출의 관점에서 커뮤니티 토픽의 사용을 소개합니다.

커뮤니티 토픽의 API 사용

1. 먼저 createGroup 인터페이스를 호출하여 토픽을 지원하는 커뮤니티를 생성하고, 커뮤니티 관리 '커뮤니티 생성' 단계를 참고하여 구현할 수 있습니다.
2. 그 다음 getJoinedCommunityList 인터페이스를 호출하여 이러한 유형의 생성 및 가입 커뮤니티 목록을 가져올 수 있습니다. 커뮤니티는 그룹 구성원을 관리하는 데 사용되지만 토픽을 지원하는 커뮤니티에서 메시지를 주고 받을 수는 없습니다. 커뮤니티의 다른 기능에 대해서는 일반 그룹 API를 참고하십시오.
3. 커뮤니티가 성공적으로 생성되면 커뮤니티에서 createTopicInfoCommunity를 호출하여 여러 개의 토픽을 생성할 수 있습니다. Topic Management의 'Creating a topic' 단계를 참고하십시오.
4. 토픽 관리에는 '토픽 삭제', '토픽 정보 수정', '토픽 목록 가져오기' 및 '토픽 콜백 수신' 기능도 포함됩니다. 동시에 토픽은 사용자가 메시지를 주고받을 수 있는 커뮤니케이션의 장이며, 관련 인터페이스는 메시지 생성 소개를 참고하

십시오.
5. 그룹 커뮤니티 구성원: 그룹 구성원 사용자 정의 필드에 그룹 정보를 지정하여 그룹의 커뮤니티 구성원을 표시할 수 있습니다. 이 효과를 구현하려면 모든 커뮤니티 구성원을 로컬로 끌어와 그룹별로 정렬해야 합니다. 그룹 구성원 수가 많은 경우 서버측에서 구성원 그룹화를 구현하는 것이 좋습니다.

관련 문서

그룹 관리
그룹 시스템
SDK 다운로드
SDK 매뉴얼
Android
SDK Integration (iOS)
SDK Integration (Web)

문의하기

사용에 문제가 있는 경우 문의하기를 통해 문의해 주시기 바랍니다.

How to integrate Tencent IM with Salesforce

최종 업데이트 날짜: : 2024-02-07 17:30:51

Introduction

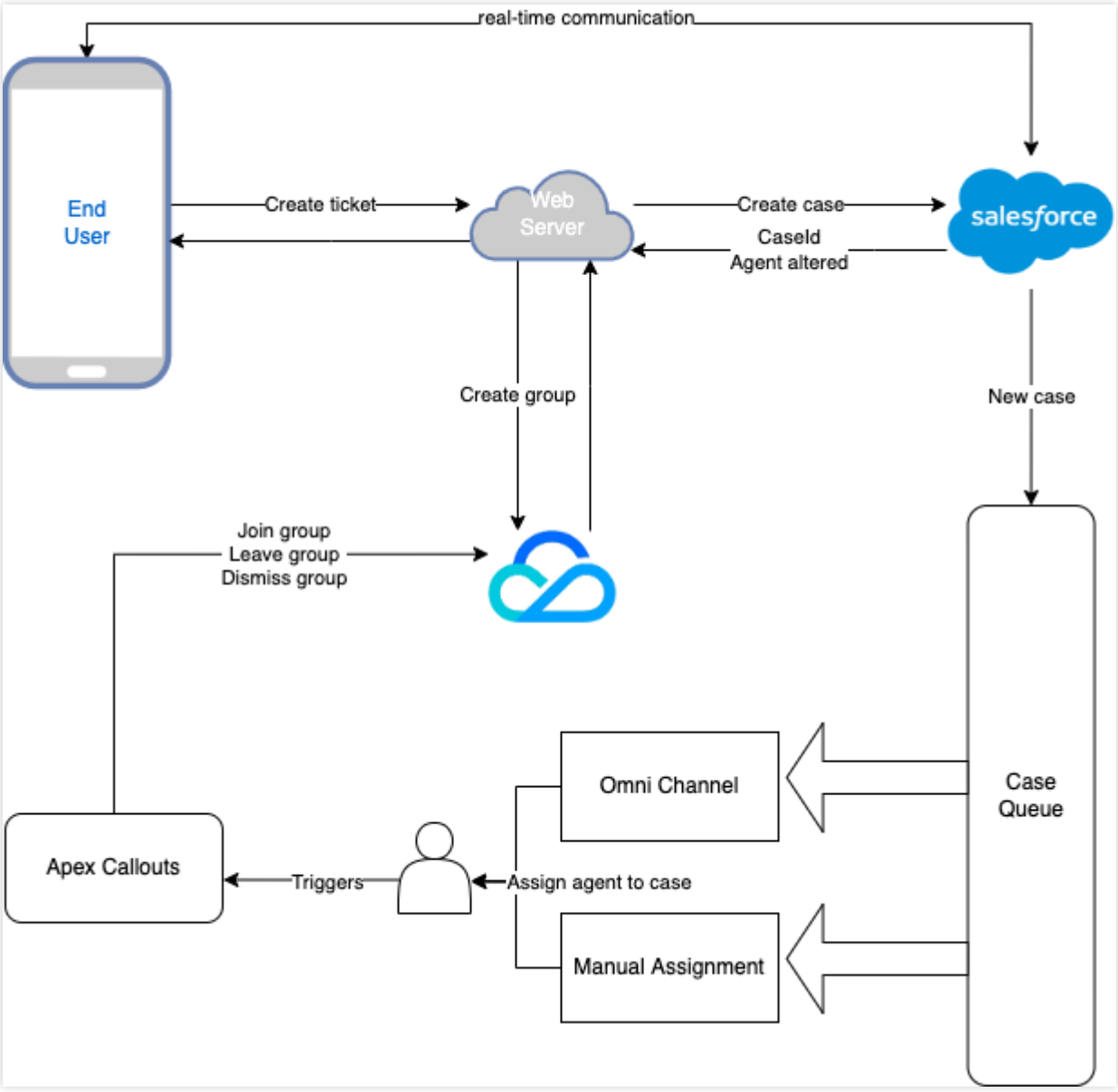
This tutorial aims to demonstrate an approach to integrate the Tencent Cloud IM SDK into Salesforce's workflow to leverage the communication between a Salesforce agent and an end user.

Preliminary

1. Sign up for Tencent Cloud, register IM service and create an app, see guidence.
2. Sign up for a Salesforce developer account in case you don't have, click here.

Road Map

Three parts are essential to achieve the goal.
1. An end user application for end user to start an conversation.
2. An online server for creating a Salesforce case and creating an Tencent Cloud IM chat group. Also provide APIs for invite/delete Salesforce Agent to the group and dismiss the group when case is closed.
3. An custom Salesforce utilities component for in-Salesforce communication.
Here's the integration map:



End User Application

Tencent Cloud IM provides a variety of SDKs for the most popular platforms, and you can simply choose the one for your platform, check out our SDKs here: Android, iOS, Web, Flutter, Windows, Unity, Unreal Engine. The recommended way to build your application from scratch is to utilize our TUIKit to layer up the interface, see here: Android, iOS, Web, Flutter.

Online Server Relay

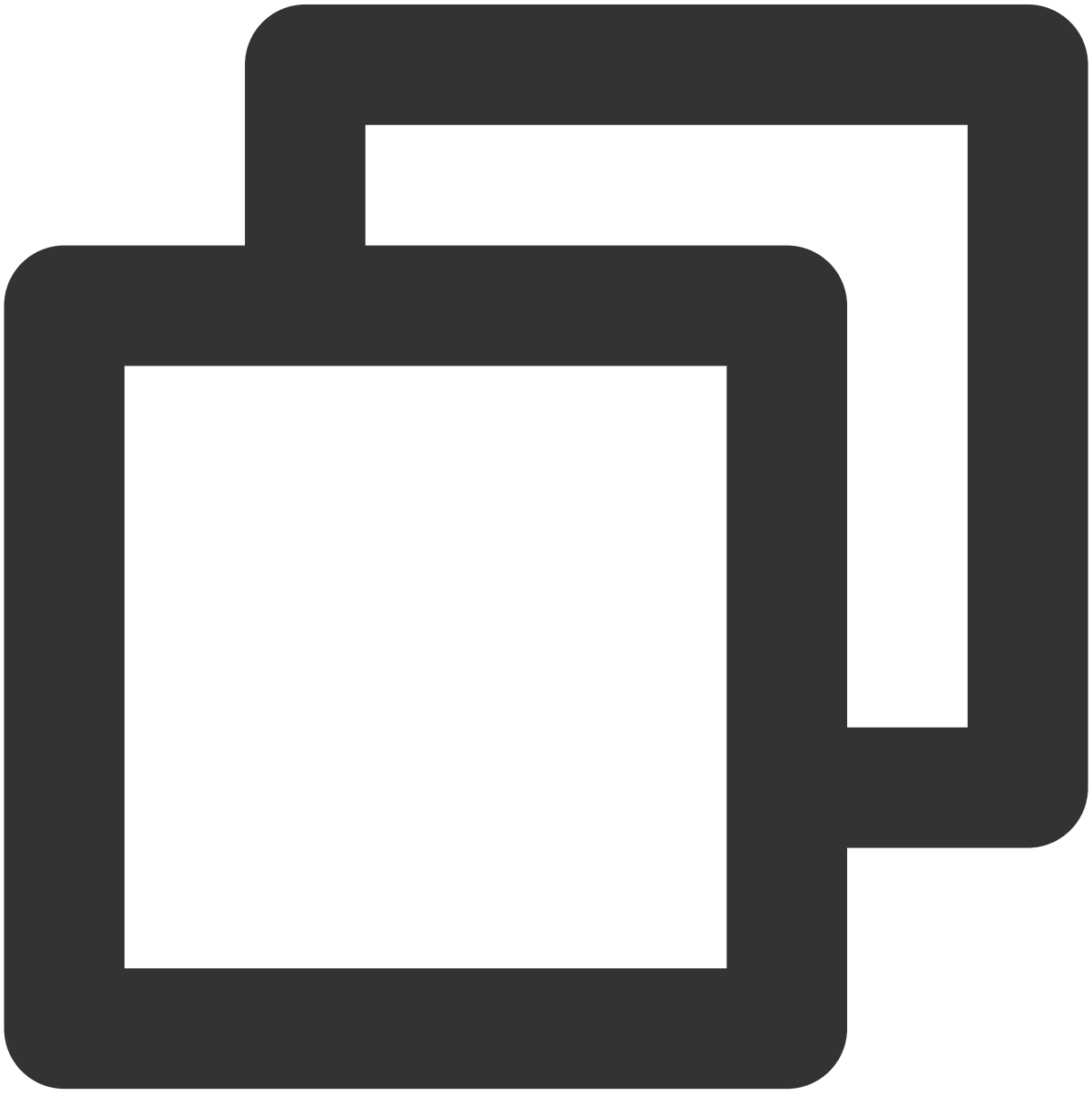
In here, we will guide you step by step to create a web server to allow an end user to submit a Salesforce case and create an Tencent Cloud IM chat group, also allow Salesforce agents to be invited/removed from Tencent Cloud IM chat group and also delete the group when case is closed.

Agent Chat Interface

This tutorial will give you the instructions to create an chat interface in Salesforce and invite agent into the Tencent Cloud IM chat group. Also, you can use our Web UIKit to build this plug-in widget.

Step 1. Create an online server

The online server is for the purpose to connect Salesforce and Tencent Cloud IM, and enables the end user to create a Salesforce case and create an corresponding Tencent Cloud IM chat group.
The example Node server is shown below:



```
const express = require("express")
const axios = require("axios")
var TLSSigAPIv2 = require("tls-sig-api-v2") // Generate UserSig for Tencent Cloud I
const sf = require("node-salesforce") // Salesforce API Connection Library for Node

const YOUR_SDKAPPID = 1400000000
const YOUR_SECRET = ""
const ADMIN_USERID = ""

const app = express()
app.use(express.json())

const port = process.env.PORT || 3000

app.use(express.json())

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "https://YOUR_DOMAIN")
  res.header("Access-Control-Allow-Headers", "*")
  next()
})

// End user calls /createticket to create a Salesforce case and a Tencent Cloud IM
app.post("/createticket", async (req, res) => {
  const { userId, caseInfo } = req.body
  if (!userId) return res.status(500).send("Missing userId")

  const auth = await getSalesforceAccessToken()
  if (auth.error) return res.status(500).send("Salesforce auth error")

  const salesforceCase = await createCase(auth.token, caseInfo)
  if (!salesforceCase.success)
    return res.status(500).send("Case creation failed")

  const groupName = salesforceCase.id
  const result = await createGroup(groupName, userId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Group creation failed")

  res.status(200).send(result)
})

// When detect a new agent assigned to the group, Salesforce sends a request to joi
app.post("/joingroup", async (req, res) => {
  const { groupId, userId } = req.body
  if (!userId) return res.status(500).send("Missing userId")
```

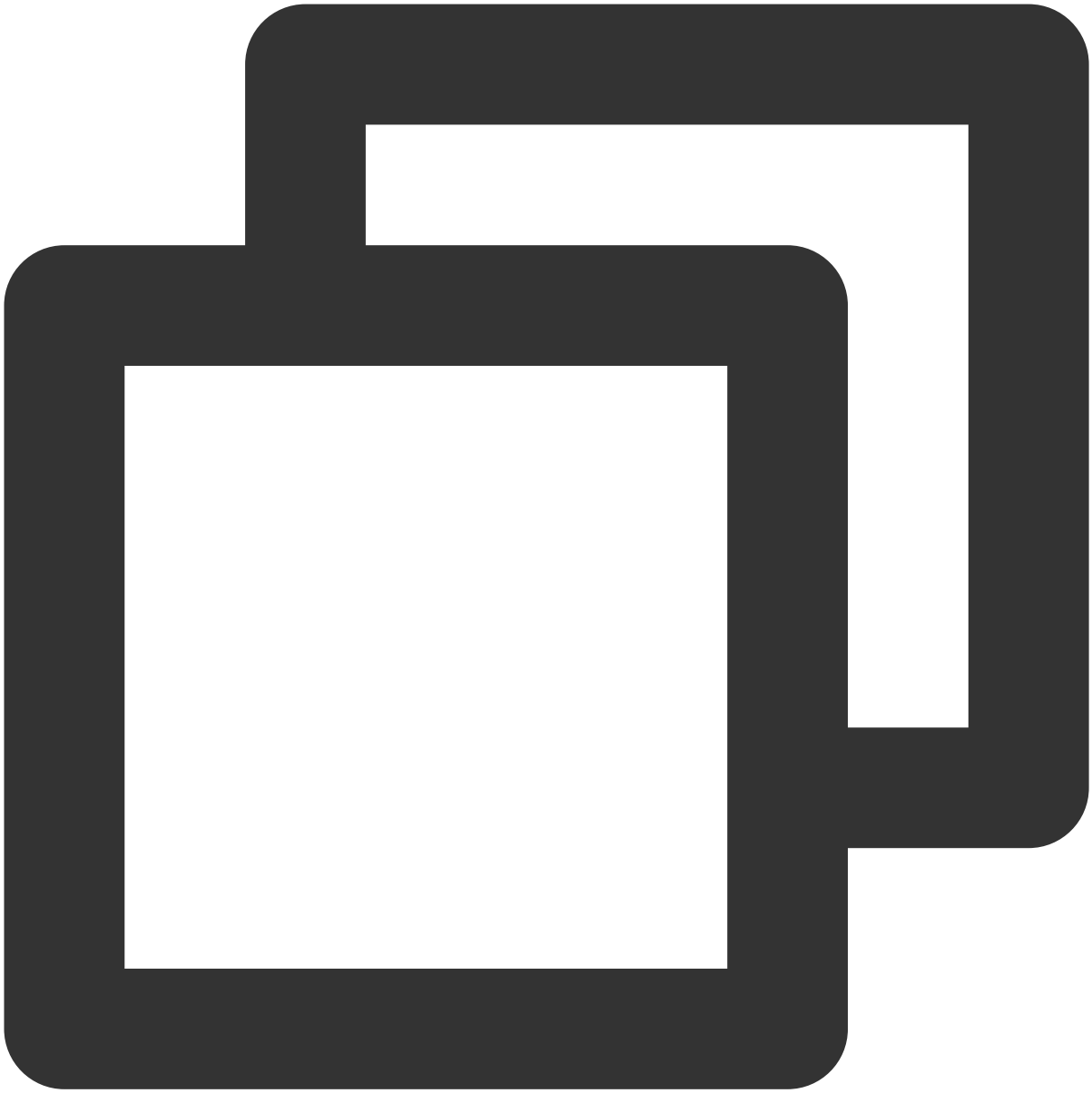
```
  if (!groupId) return res.status(500).send("Missing groupId")

  const result = await joinGroup(groupId, userId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Join group failed")

  res.status(200).send(result)
})

// When detect an agent removed from the group, Salesforce sends a request to remov
app.post("/leavegroup", async (req, res) => {
  const { groupId, userId } = req.body
  if (!userId) return res.status(500).send("Missing userId")
  if (!groupId) return res.status(500).send("Missing groupId")

  const result = await leaveGroup(groupId, userId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Leave group failed")

  res.status(200).send(result)
})

app.post("/deletegroup", async (req, res) => {
  const { groupId } = req.body
  if (!groupId) return res.status(500).send("Missing groupId")

  const result = await deleteGroup(groupId)
  if (result.ErrorCode !== 0)
    return res.status(500).send("Delete group failed")

  res.status(200).send(result)
})

const getSalesforceAccessToken = async function () {
  const url = "https://{your_instance}.salesforce.com"
  const conn = new sf.Connection({ loginUrl: url })
  try {
    await conn.login("SF_EMAIL", "SF_PASSWORDSF_TOKEN")
    return { error: undefined, token: conn.accessToken }
  } catch (e) {
    return { error: e, token: undefined }
  }
}

const createCase = async function (token, caseInfo) {
  const { subject, desc, name, email } = caseInfo
  const body = {
```

```
    Subject: subject,
    Description: desc,
    SuppliedName: name,
    SuppliedEmail: email,
  }
  const headers = {
    headers: {
      "Content-Type": "application/json",
      Authorization: "Bearer " + token,
    },
  }
  const url =
    "https://{your_instance}.salesforce.com/services/data/v{api_version}/sobjects/C
  try {
    const result = await axios.post(url, body, headers)
    return result.data
  } catch (e) {
    return { id: undefined, success: false, error: e }
  }
}

const generateUserSig = function () {
  const expires = 600
  const api = new TLSSigAPIv2.Api(YOUR_SDKAPPID, YOUR_SECRET)
  return api.genSig(ADMIN_USERID, expires)
}

const generateRandom = function () {
  return Math.floor(Math.random() * 4294967295)
}

const createGroup = async function (groupName, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  // Use salesforceCase.id as group ID
  const data = { Owner_Account: userId, Type: "Public", Name: groupName, GroupId: g
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

const joinGroup = async function (groupId, userId) {
  const sig = generateUserSig()
```

```
  const random = generateRandom()
  const data = { GroupId: groupId, MemberList: [{ Member_Account: userId }] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?s
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

const leaveGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberToDel_Account: [userId] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/delete_group_membe
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

const deleteGroup = async function (groupId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?sdka
  try {
    const groupRes = await axios.post(url, data)
    return groupRes.data
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e }
  }
}

app.listen(process.env.PORT || port, () =>
  console.log(`Example app listening on port ${port}!`)
)
```

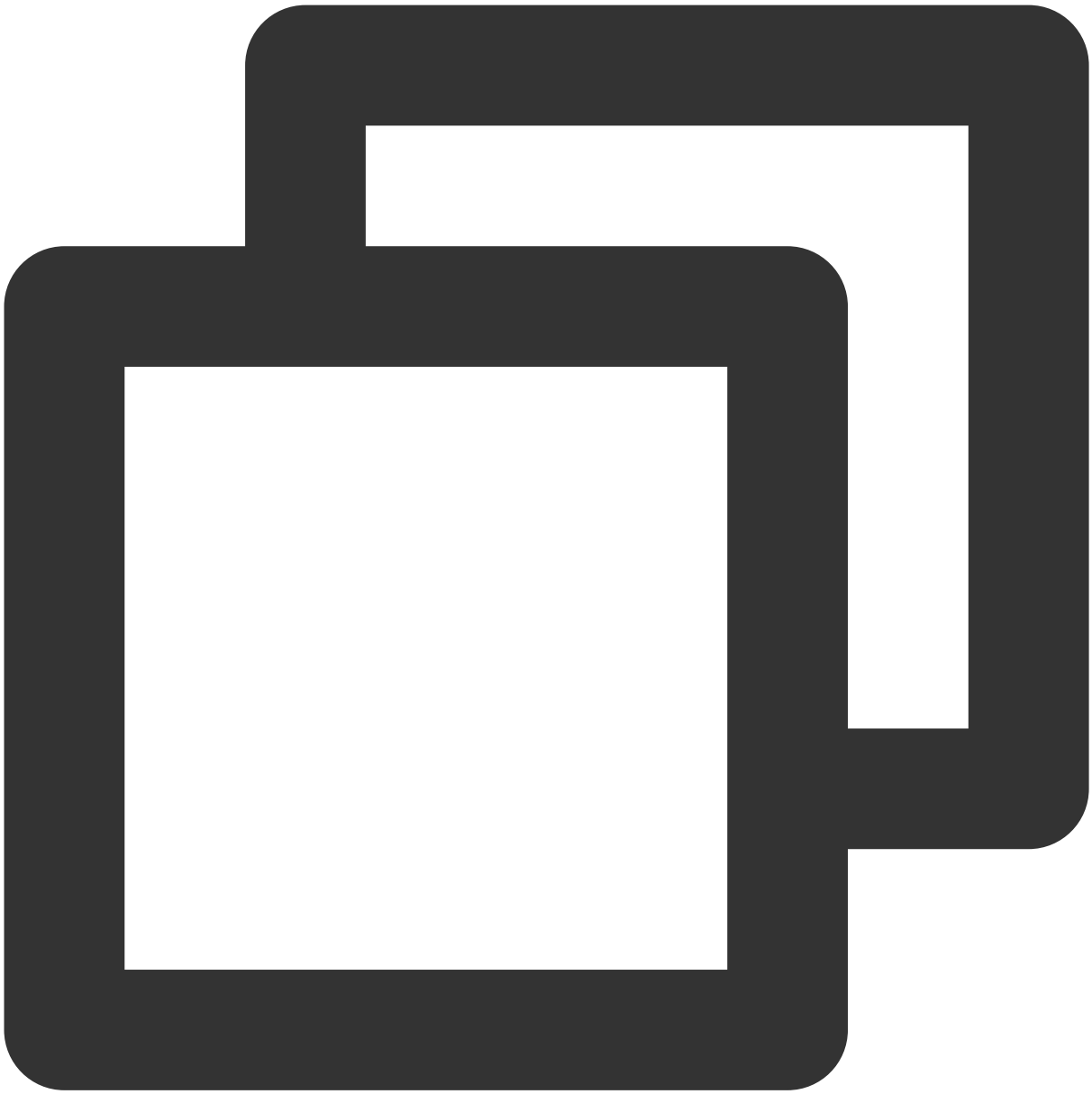
The server supports a route /createticket for the end user to create a case in Salesforce and a chat group in Tencent Cloud IM. Here's what we do here:

1. Fisrt we fetch an accessToken from Salesforce, more about SF_TOKEN.



```
const getSalesforceAccessToken = async function () {
  const url = "https://{your_instance}.salesforce.com"
  const conn = new sf.Connection({ loginUrl: url })
  try {
 await conn.login("SF_EMAIL", "SF_PASSWORDSF_TOKEN")
 return { error: undefined, token: conn.accessToken }
  } catch (e) {
 return { error: e, token: undefined }
  }
}
```

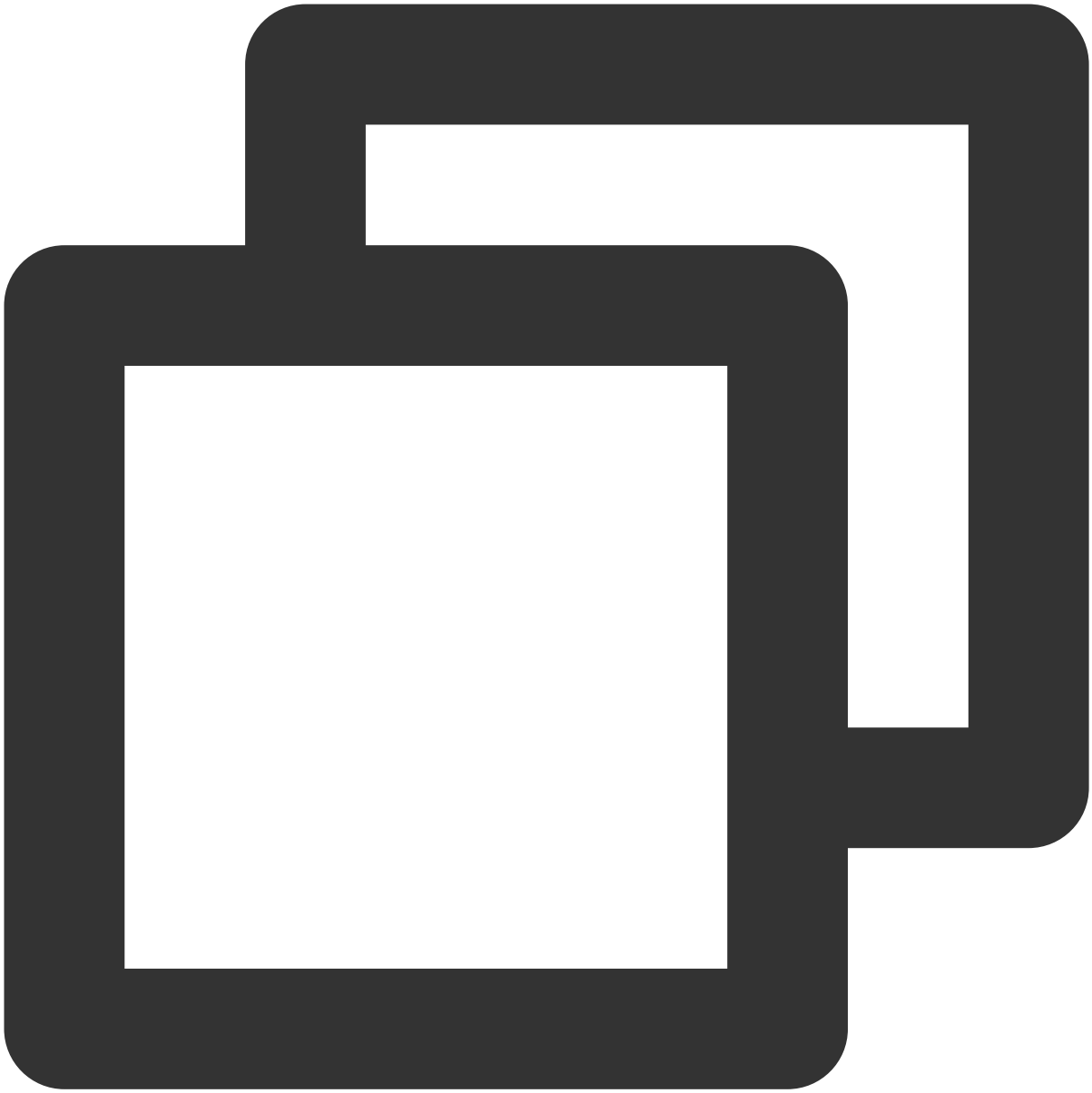
2. Create a Salesforce case.



```
const createCase = async function (token, caseInfo) {
  const { subject, desc, name, email } = caseInfo
  const body = {
 Subject: subject,
 Description: desc,
 SuppliedName: name,
 SuppliedEmail: email,
  }
  const headers = {
```

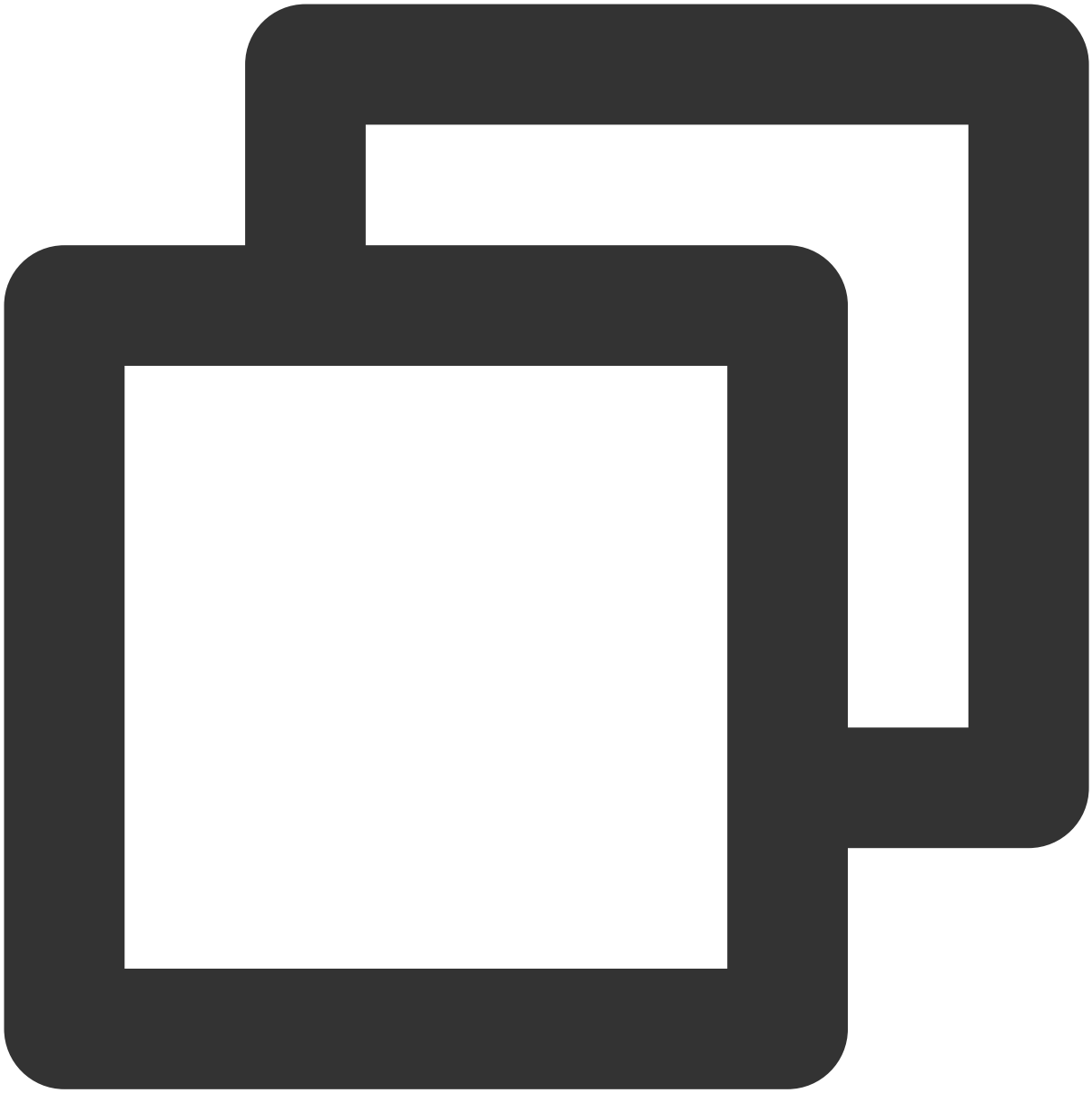
```
  headers: {
    "Content-Type": "application/json",
    Authorization: "Bearer " + token,
  },
   }
   const url =
  "https://{your_instance}.salesforce.com/services/data/v{api_version}/sobjects/Case
   try {
  const result = await axios.post(url, body, headers)
  return result.data
   } catch (e) {
  return { id: undefined, success: false, error: e }
   }
}
```

3. Generate UserSig for Tencent Cloud IM



```
const generateUserSig = function () {
  const expires = 600
  const api = new TLSSigAPIv2.Api(YOUR_SDKAPPID, YOUR_SECRET)
  return api.genSig(ADMIN_USERID, expires)
}
```

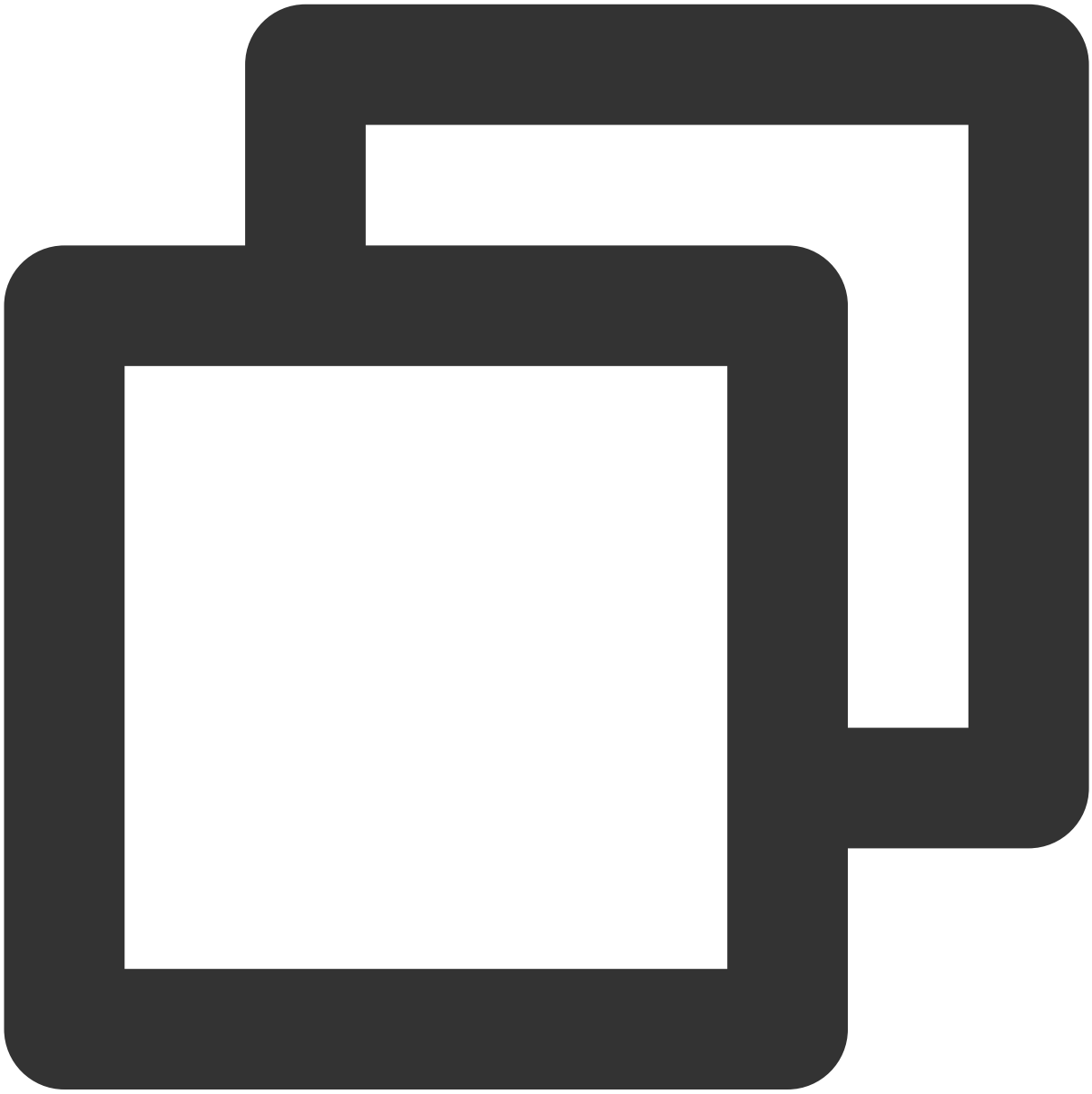
4. Create a Tencent Cloud IM chat group by case ID and user ID



```
const createGroup = async function (groupName, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { Owner_Account: userId, Type: "Public", Name: groupName }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
```

```
}
```

In addition, the web server provides routes to join/leave/delete a Tencent Cloud IM chat group by case ID and agent ID. These are used when Salesforce trigger detects the changing of case agent. More details will be discussed in Step 3.



```
const joinGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberList: [{ Member_Account: userId }] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?s
```

```
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
}

const leaveGroup = async function (groupId, userId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId, MemberToDel_Account: [userId] }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/delete_group_membe
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
}

const deleteGroup = async function (groupId) {
  const sig = generateUserSig()
  const random = generateRandom()
  const data = { GroupId: groupId }
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?sdka
  try {
 const groupRes = await axios.post(url, data)
 return groupRes.data
  } catch (e) {
 return { ErrorCode: -1, ErrorInfo: e }
  }
}
```

That's all for the web server side. Once you set up the server, call the endpoint /createticket and

Check in Salesforce that a case is created.

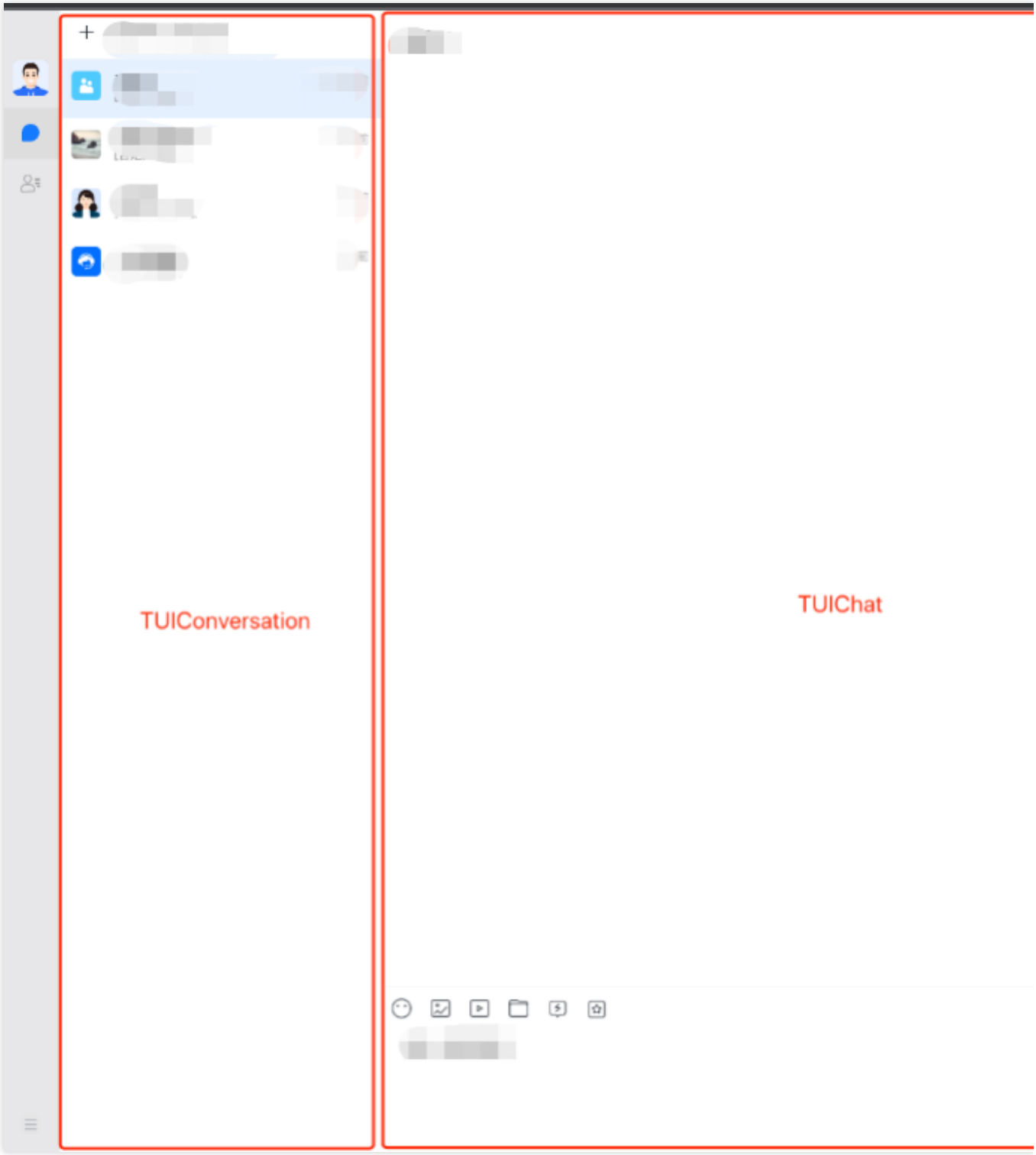
Check in the Tencent Cloud IM console for the group with the case ID as the group ID.

Step 2. Use the Tencent Cloud IM Web UI Kit to build a Salesforce utilities component

Here we show the steps to create a Salesforce utilities component with Tencent Cloud IM UI Kit. In Salesforce you may use Lightning Container to upload a third-party i-frame as a static resource, and host the content in an Aura

component using `lightning:container` . And you can use Tencent Cloud IM Web UIKit to build an agent chat component, and deploy it in the Lightning Container as a Salesforce utilities bar widget at the bottom.

1. First develop a chat component by using Tencent Cloud IM Web UIKit. Build it as a static resource with a root index.html and compress it as an zip file. Case agent's ID will be transmitted to the chat component, use that ID to init and login Tencent Cloud IM in the chat component.



2. Create a Lightning Container for your component, details arehere.

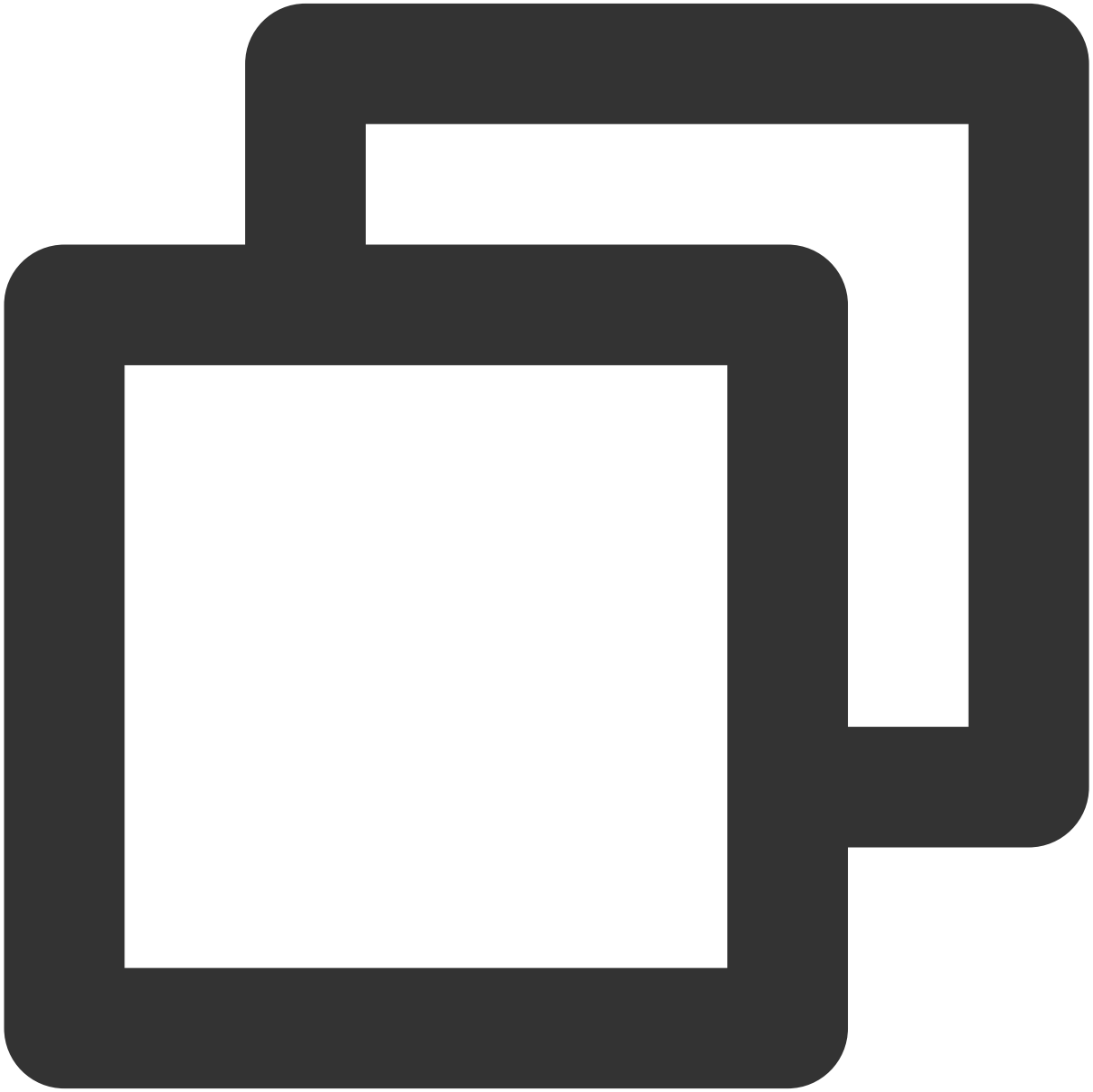
2.1 Go to Salesforce Developer Console

2.2 Click File -> New -> Lightning Component

2.3 Name = "tim_utilities_bar"

2.4 Click submit

3. Render the custom component to your bar widget. 3.1 Set aura:component as a utility bar and provide an aura:id



```
<!-- tim_utilities_bar.cmp -->
```

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
  <lightning:utilityBarAPI aura:id="utilitybar" />
</aura:component>
```

3.2 Upload the static resource to the Lightning Container

a. Go to Salesforce static resources and create a new resource called "tim_bar"

b. Upload the zip file of the resource and set "Cache Control" to "Public"

c. Click Save

Notes:

Index.html should always be at the root level of the .zip file

Be sure to click “save” after uploading your file.

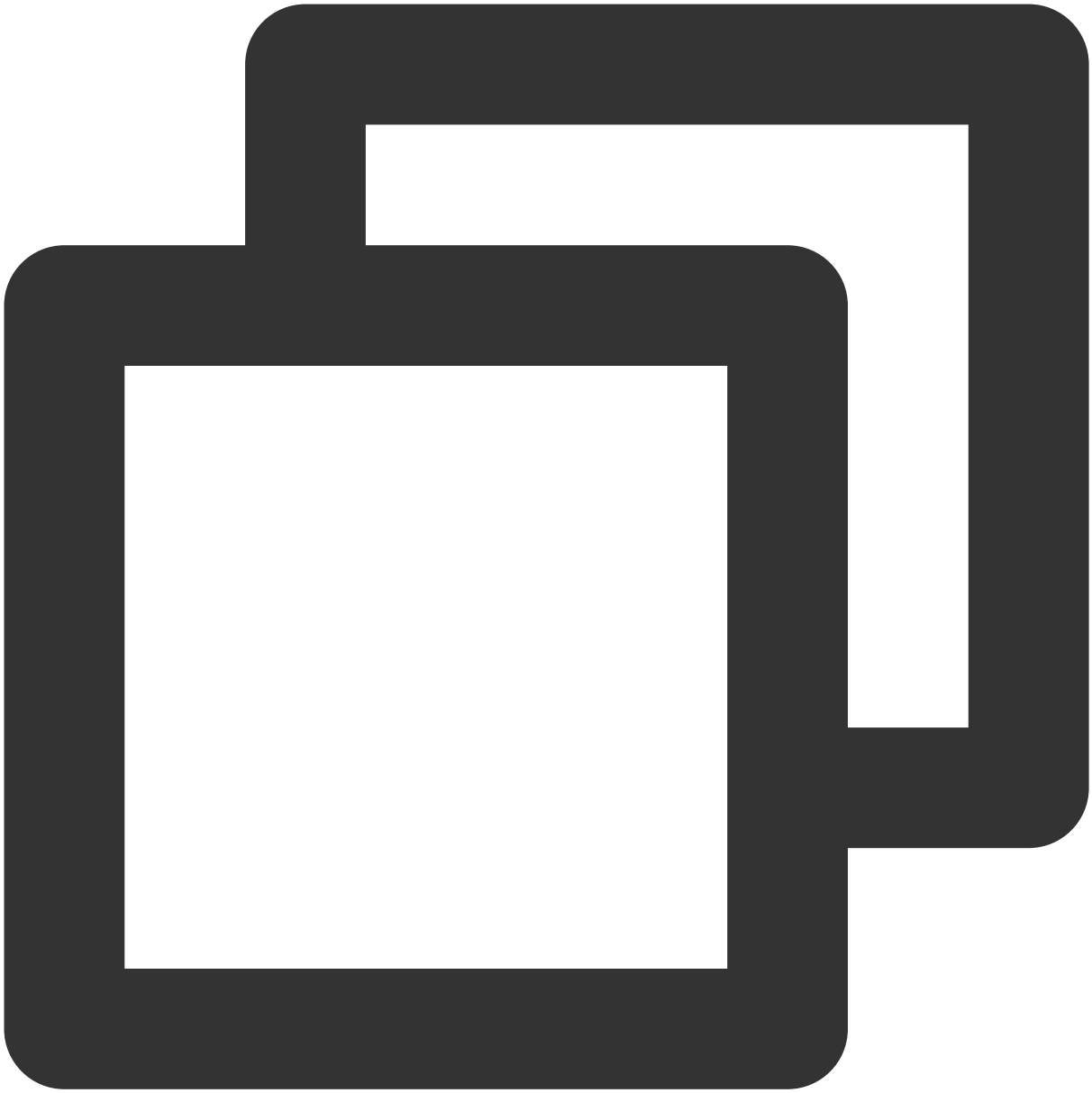
Salesforce saves the resource name, not the name of the .zip file.

100% of the code and assets you use in a Lightning Component will need to be included in the .zip file. Any external code dependencies will not work even if they are whitelisted in the CSP trusted sites list.

3.3 Reference the static resource "tim_bar" in the Utilities Bar widget

a. Add a lightning:container tag to the Utilities Bar widget. The aura:id should be "TIM_Bar".

b. Reference the static resource. "!$Resource.tim_bar + '/index.html'}" . Note that tim_bar is the saved "static resource", not the name of the uploaded .zip file.



```
<!-- tim_utilities_bar.cmp -->
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
  <lightning:utilityBarAPI aura:id="utilitybar" />
  <aura:attribute name="recordId" type="String" />
  <aura:attribute name="data" type="String" />
  <lightning:navigation aura:id="navService" />
  <lightning:container
    aura:id="TIM_Bar"
    src="{!$Resource.tim_bar + '/index.html'}"
  />
</aura:component>
```

3.4 Add the Utilities Bar Widget to display in Salesforce

a. Click "Setup" and search for "App Manager" to set where the Utilities Bar widget will appear

b. Click "▾" and "Edit" in the App called Service Console

c. In App Setting, click "Utility Items (Desktop Only)"

d. Click "Add Utility Item"

e. Select the "tim_utilities_bar"

f. Set the width and height

g. Check "Start automatically"

h. Click -> "Save"

3.5 Update the Salesforce CSP file to grant permissions to Access Tencent Cloud IM in Salesforce

a. Go to Salesforce -> Setup -> Search -> "CSP Trusted sites"

b. Add "New Trusted Sites" (allow all CSP Directives):

wss://wss.im.qcloud.com

c. Go to Salesforce -> Setup -> Search -> "CORS"

d. Add "New" Allowed Origins List:

https://*.qq.com

https://*.qcloud.com

e. Go to Salesforce -> Setup -> Search -> ""

f. Add "New Remote Site" List:

The Url of the web server!

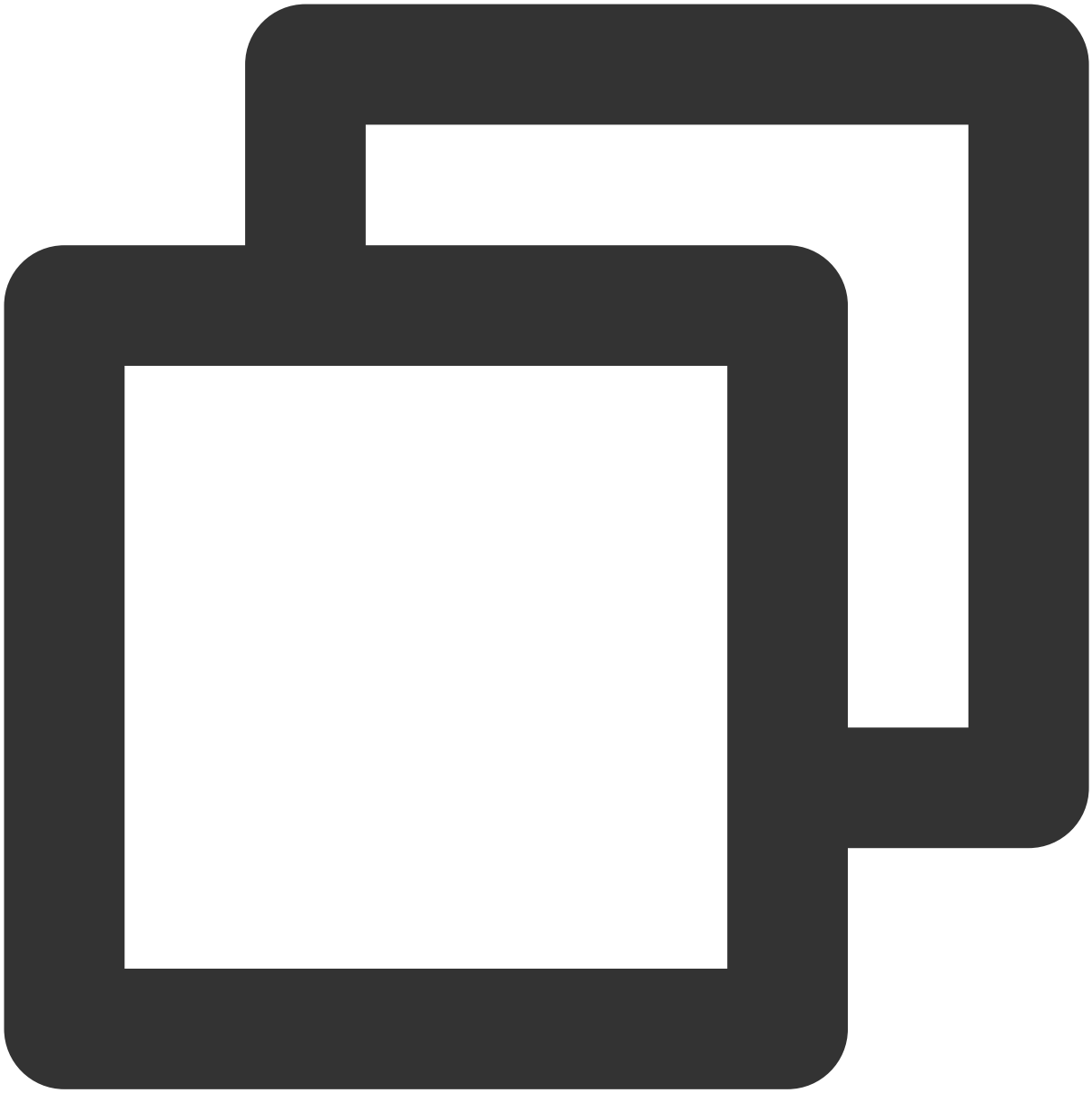
4. Initialize the Lightning Container

Once Lightning Container is ready, send an LLC message to inform Utilities Bar Widget

Utilities Bar Widget needs to send Agent's id to our component

Once received messages from Utilities Bar Widget, we render the UIKit

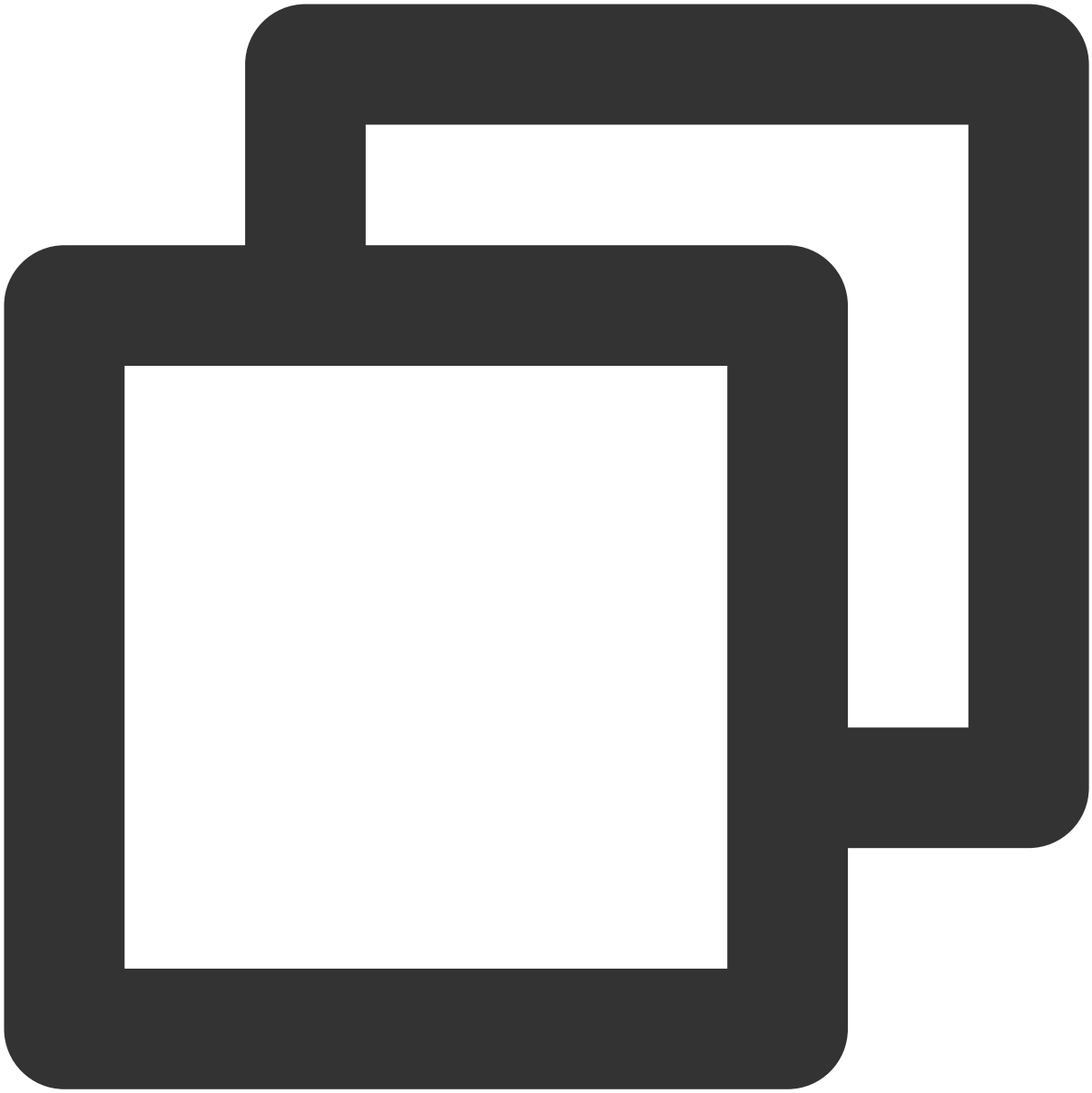
Follow the code shown below:



```
// tim_utilities_barController.js
({
handleMessage: function(component, message, helper) {
var payload = message.getParams().payload
// Once container is ready, initUIKit
if (payload === "READY") helper.initUIKit(component, message, helper)
}
});
({
initUIKit: function (component, message, helper) {
// Get Agent's ID
```

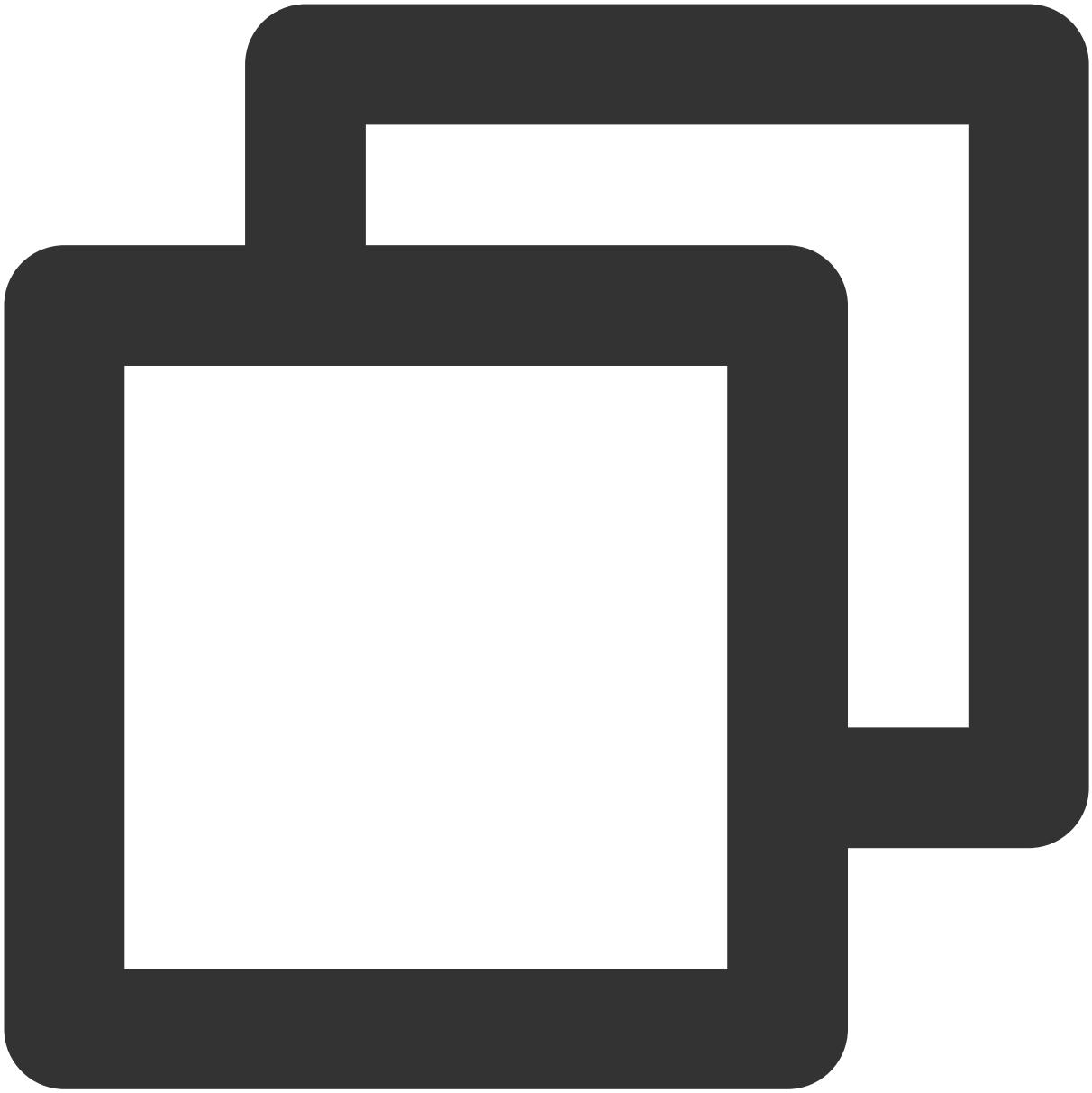
```
var userId = $A.get("$SObjectType.CurrentUser.Id")
var message = { userId: userId }
try {
// Send the ID to the component
component.find("TIM_Bar").message(message)
} catch (err) {
console.error("Error from Utilities Bar:", err)
}
},
})
```

Add onMessage handler to the Utilities Bar Widget：



```
<!-- tim_utilities_bar.cmp -->
<lightning:container
aura:id="TIM_Bar"
src="{!$Resource.tim_bar + '/index.html'}"
onmessage="{!c.handleMessage}"
/>
```

In your script, use LLC package to render your app when Lightning Container has loaded.



```
// index.js
try {
const clientState = "READY";
LLC.sendMessage(clientState);
console.warn("Lightning Container --> TO SALESFORCE --> Sent:", clientState);
} catch (e) {
console.error("LLC NOT WORKING", e);
}
try {
LLC.addErrorHandler((error) => console.log("LLC ERROR:", error));
LLC.addMessageHandler((salesforceMessage) => {
```

```
console.warn("SALESFORCE --> Lightning Container --> Arrived:", salesforceMessage);
const app = createApp(App, {
user: salesforceMessage
});
app
.use(store)
.use(router)
.use(TUIKit)
.use(Aegis)
.use(ElementPlus)
.mount('#app');
});
} catch (e) {
console.error("Error from LLC!!", e);
}
```

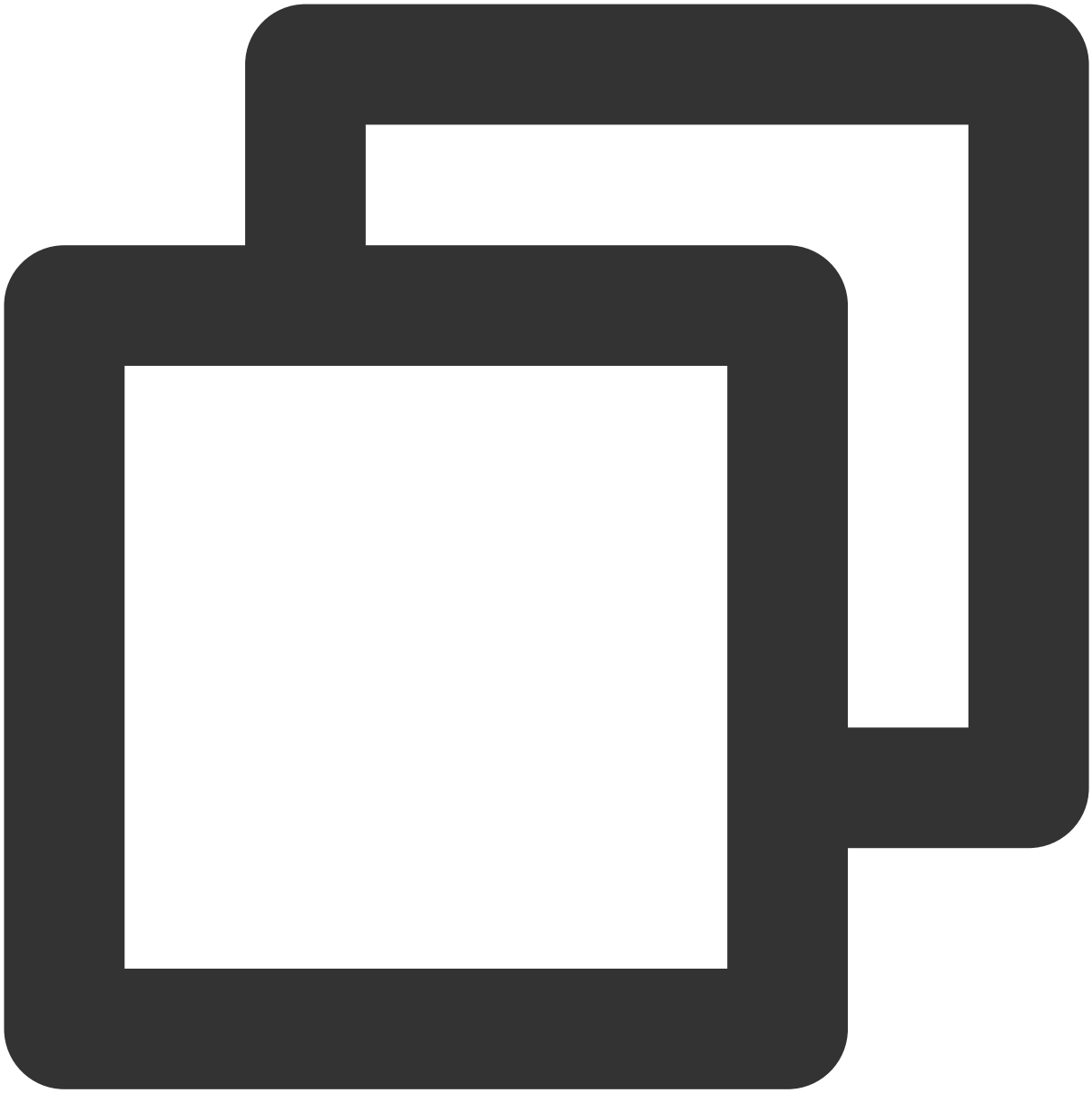
Step 3. Listen for Salesforce Case assignment and set IM Chat Group members

In Salesforce, a case is assigned to an agent manually or automatically. Hence we need to invite new agent to the group and make the previous agent leaves the group. We use Salesforce Apex Callous to listen for the changing of the case agent assignment. Here's what to do by calling the Salesforce Apex Callout.

1. Listen to manual case assignment

When the designated agent to one case is changed, the Apex Case Change Trigger calls the Apex Callout. In the callout, we a. delete the previous agent from the Tencent Cloud IM chat group and b. invite the new agent to the group. And when case is deleted, dismiss the Tencent Cloud IM group accordingly.

Go to Salesforce Developer console –> New –> Apex Trigger –> Name = "AssignAgent" & sObject = "Case"



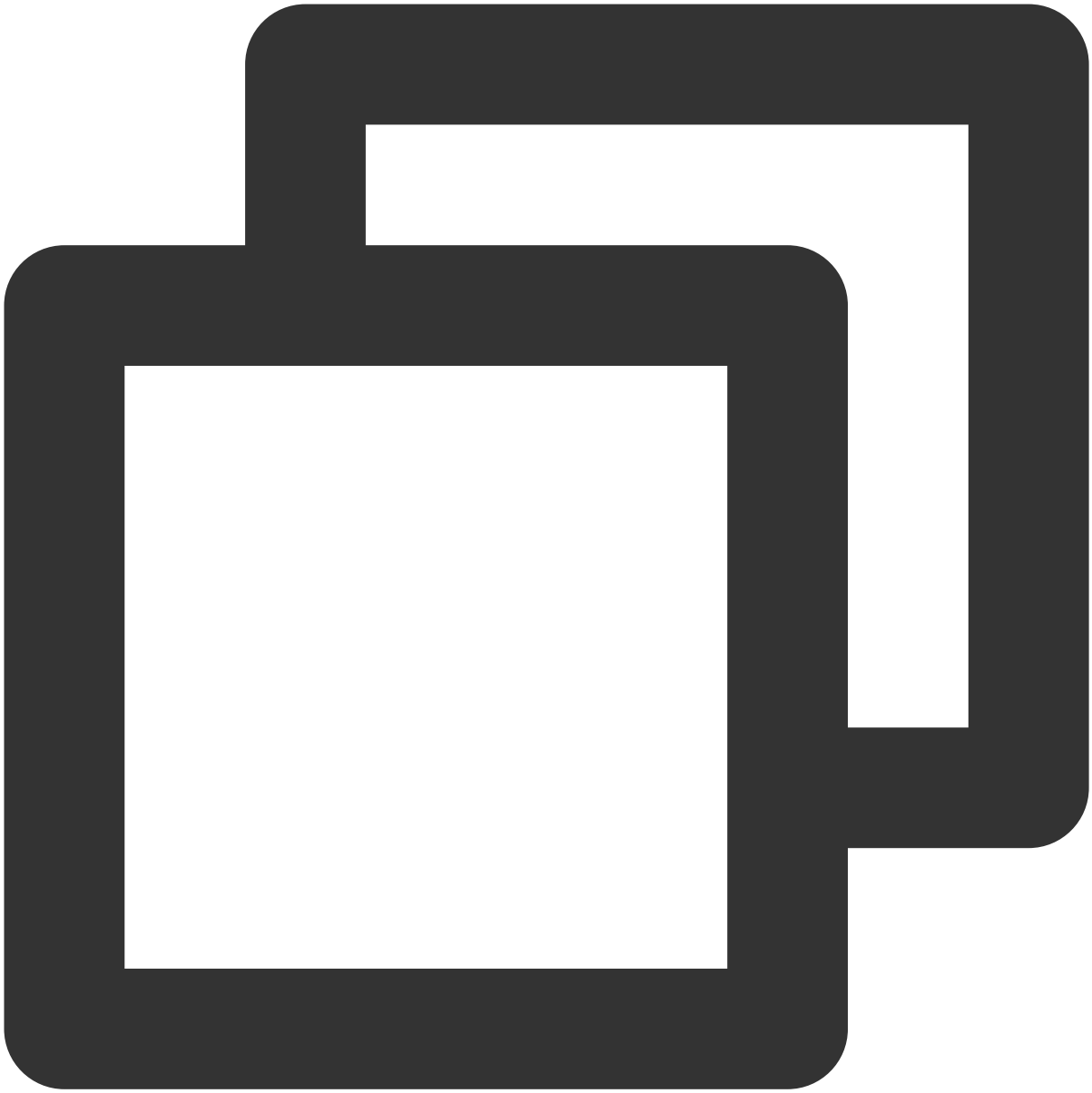
```
// AssignAgent.apxt
trigger AssignAgent on Case (after update, after delete) {
     if(trigger.isUpdate){
       // Case is updating
       System.debug('Case Update Fired:');
       for(Case a : trigger.new){
         Case oldCase = trigger.oldMap.get(a.ID);
         if(String.valueOf(a.OwnerId).substring(0, 3) == '005'){
           // Owner Agent ID is changed, 005 prefix means agent ID
           System.debug('Agent invited :' + a.OwnerId);
           // Assign new owner to the Tencent Cloud IM group
```

```
            String[] data = new String[2];
            data[0] = a.Id; // Case ID is group ID
            data[1] = a.OwnerId;
            TimCallouts.joinGroup(data); //Custom callout class
          }
          // New case agent is different fromthe current agent
          if(String.valueOf(oldCase.OwnerId).substring(0, 3) == '005' && oldCase.Own
            // Delete the old agent from the group.
            // Note: A Case's very first owner will be the system owner.
            System.debug('Old Agent will be removed from group' + a.Id);
            System.debug('leaveGroup: ' + oldCase.OwnerId);
            String[] removeData = new String[2];
            removeData[0] = a.Id; // Case ID is group ID
            removeData[1] = oldCase.OwnerId;
            TIMCallouts.leaveGroup(removeData);
          }
        }
    }
    if(trigger.isDelete ){
      // Case is deleting
      System.debug('Case Delete Fired:');
      for(Case a : trigger.old){
        if(String.valueOf(a.OwnerId).substring(0, 3) == '005'){
          System.debug('Delete Group :' + a.Id);
          String[] data = new String[1];
          data[0] = a.Id;
          TimCallouts.deleteGroup(data); //Custom callout class
        }
      }
    }
  }
```

2. Listen to automatic case assignment by Salesforce Omni Channel

Omni Channel detects a case assignment and Salesforce automatically creates an AgentWork object. If an agent accepts the assignment, the AgentWork Trigger may use Salesforce Callout to join the group.

Go to Salesforce Developer console -> New -> Apex Trigger -> Name = "AgentOmniChannel" & sObject = "AgentWork"

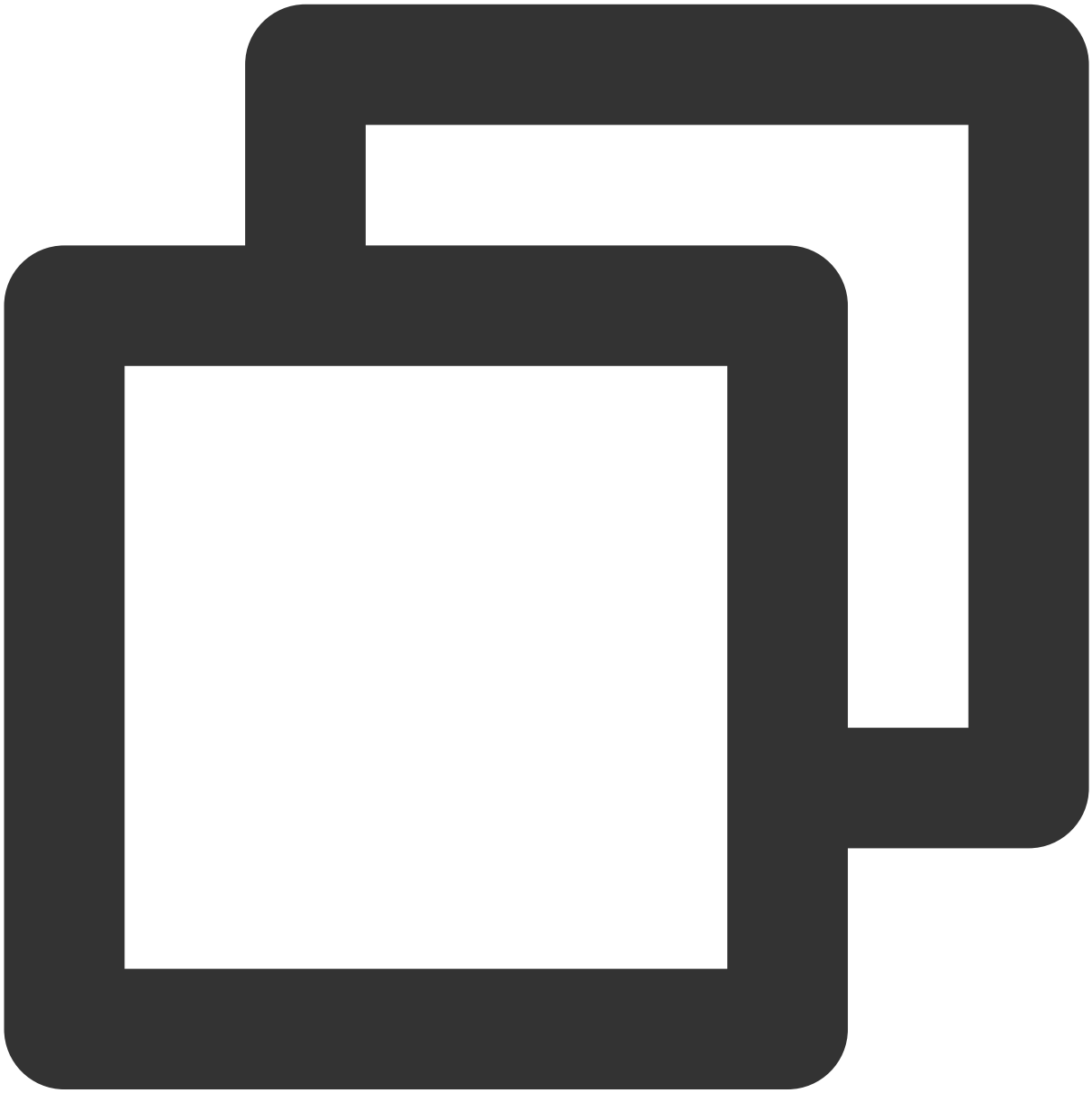


```
// AgentOmniChanne.apxt
trigger AgentOmniChannel on AgentWork (after update, after insert) {
    if(Trigger.isUpdate){
      for(AgentWork a : Trigger.new){
            AgentWork oldCase = Trigger.oldMap.get(a.ID);
            if(a.Status == 'Opened' && String.valueOf(a.OwnerId).substring(0, 3) ==
              String[] data = new String[2];
              data[0] = a.WorkItemId;
              data[1] = a.OwnerId;
              TIMCallouts.joinGroup(data);
            }
```

```
        }
    }
}
```

3. Set Salesforce Callout to invite/remove agents.

Go to Salesforce Developer console –> New –> Apex Class –> Name = "TIMCallOuts"



```
// TIMCallOuts.apxc
public class TIMCallouts {
  @future(callout=true)
  public static void joinGroup(String[] data) {
```

```
    String groupId = data[0];
    String userId = data[1];
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://{your_web_server}/joingroup');
    request.setMethod('POST');
    request.setHeader('Content-Type', 'application/json;charset=UTF-8');
    request.setBody('{"userId":"'+ userId +'", "groupId":"'+ groupId +'"}');
    HttpResponse response = http.send(request);
    // Parse the JSON response
      if (response.getStatusCode() != 200) {
        System.debug('Join group failed: '+response.getStatusCode()+' '+response.ge
      } else {
        System.debug('Tencent Cloud IM Response: ' + response.getBody());
      }
  }

  @future(callout=true)
  public static void leaveGroup(String[] data) {
      String groupId = data[0];
      String userId = data[1];
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://{your_web_server}/leavegroup');
      request.setMethod('POST');
      request.setHeader('Content-Type', 'application/json;charset=UTF-8');
      // Set the body as a JSON object
      request.setBody('{"userId":"'+ userId +'", "groupId":"'+ groupId +'"}');
      HttpResponse response = http.send(request);
      // Parse the JSON response
      if (response.getStatusCode() != 200) {
        System.debug('Leave group failed: ' + response.getStatusCode() + ' ' + resp
      } else {
        System.debug('Tencent Cloud IM Response: ' + response.getBody());
      }
    }

    @future(callout=true)
    public static void deleteGroup(String data) {
      String groupId = data;
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('http://{your_web_server}/deletegroup');
      request.setMethod('POST');
      request.setHeader('Content-Type', 'application/json;charset=UTF-8');
      // Set the body as a JSON object
      request.setBody('{"groupId":"'+ groupId + '}');
```

```
        HttpResponse response = http.send(request);
        // Parse the JSON response
        if (response.getStatusCode() != 200) {
          System.debug('Delete group failed: ' + response.getStatusCode() + ' ' + res
        } else {
          System.debug('Tencent Cloud IM Response: ' + response.getBody());
        }
      }
  }
```

Conclusion

That's all you need to know to allow end users from any appliction to start chatting with a Salesforce agent. If you have any further questions please send an e-mail at tencentcloud_im@tencent.com, we'd be delighted to give you more details about the solution or any other solutions to build a modern real-time communication system via Tencent Cloud IM.

How to integrate Tencent IM with Zendesk

최종 업데이트 날짜: : 2024-02-07 17:30:51

Introduction

Zendesk, one of the most prevalent SaaS products in customer support, sales, and other customer communications. Meanwhile, according to Gartner, Tencent Cloud Instant Messaging made the champion in Chinese market and one of the most compotent providers in Communication Platform as a Service (CPaaS) in the global market. Naturally, it comes to us to bring you the solution of integrating Tencent Cloud IM with Zendesk.
In this essay, we will discuss how to build an client-agent-real-time communication system that extends the boundary of support team in Zendesk and enables support team to chat with clients in any platforms. We have accomplished Tencent Cloud IM App for Zendesk ticket bar and all you need to do is to install it as well as publish your own client side by the following instructions.
If that's not the integrate solution on your interest list, don't rush to the close button. Go through the essay and build your own private App for Zendesk by TUIKit, which is not a painstaking errand on your ticket list but instead a few man hours or days depending on the complexity of the App you want to build.

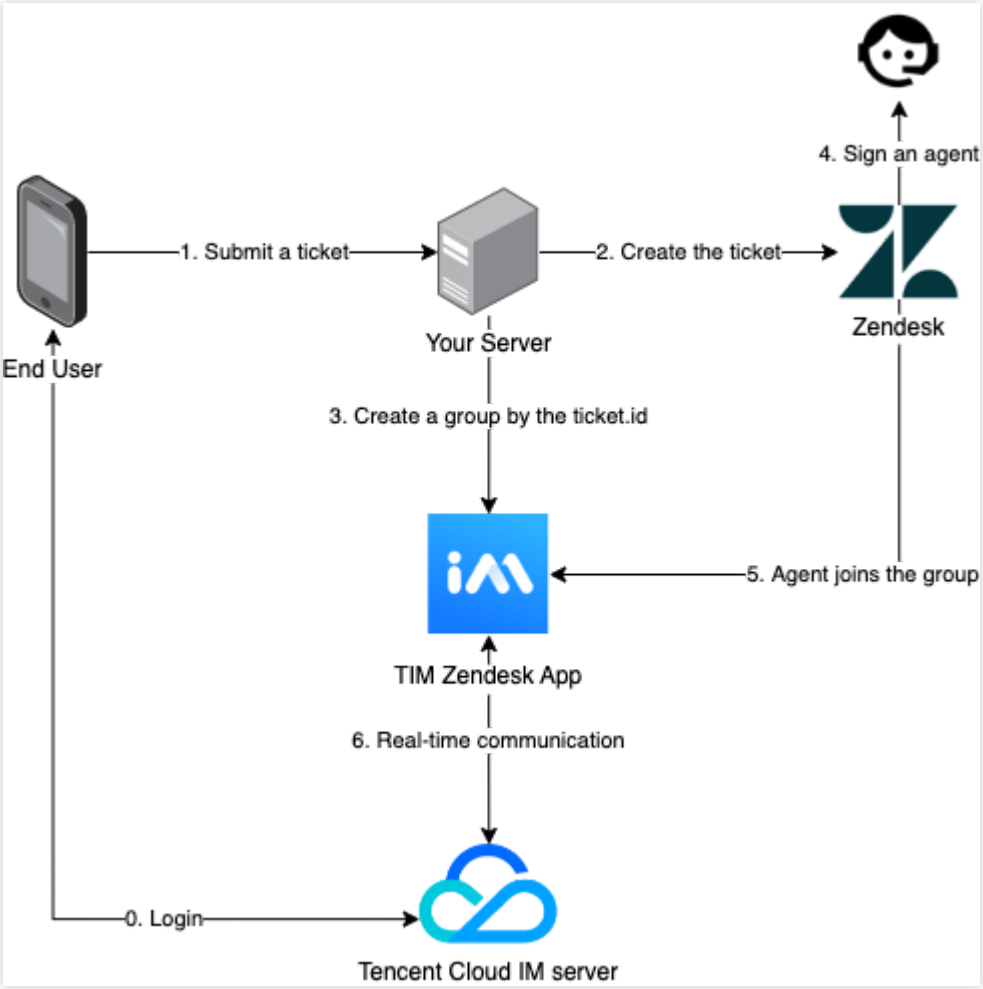
Prerequisites

If you haven't registered for Zendesk, try free trail.
To use any service of Tencent Cloud IM, you must register for Tencent Cloud, which contains so many cloud-based services other than IM. After that, click `Create Application` on the IM Console to get your `SDKAppID` , that's what you need to initiate an IM service.
For client side construction, you'll need a web server as well, which will not be included in this article.

Integration Map

The goal of this integrate soluion is straight and clear: listen to the assignment of agent and invite the agent to the conversation with the client outside Zendesk to discuss the issue or consultation in the ticket.

Here is the general integration map:



The general workflow is:

1. End user logs in to Tencent Cloud IM.

1. End user submits a ticket to your backend server.

2. According to the submitted information, create the ticket for Zendesk.

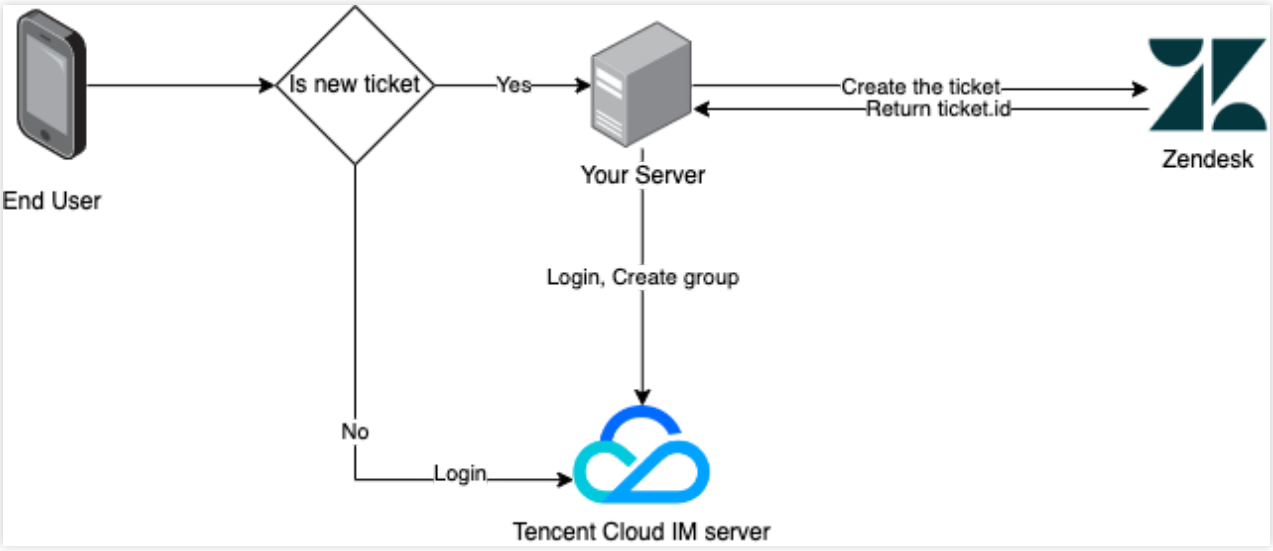
3. Create a group in Tencent Cloud IM, waiting for agent to join the group.

4. An agent takes the ticket or it is assigned to an agent.

5. The assignee joins the Tencent Cloud IM group.

6. Now client and agent can chat freely.

The workflow for the client side is:

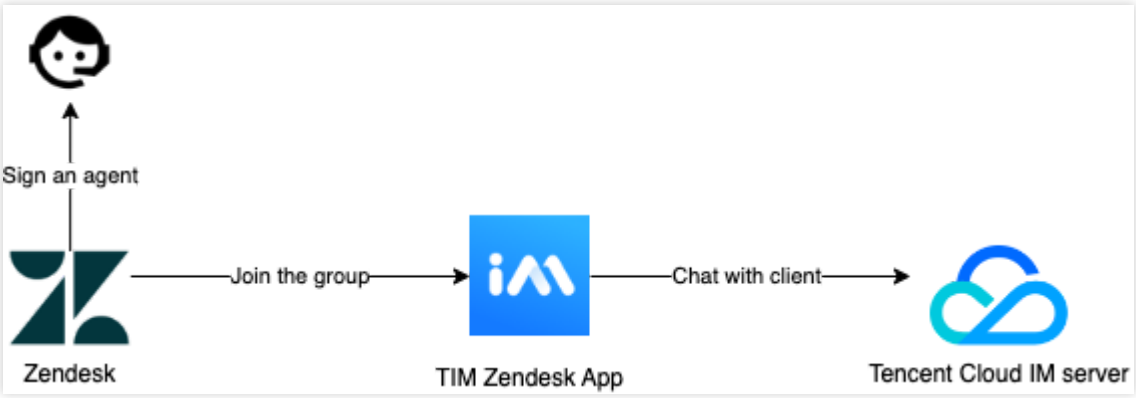


Before logged in, client may choose to reuse the ticket created last time. And for that, just login to Tencent Cloud IM server.

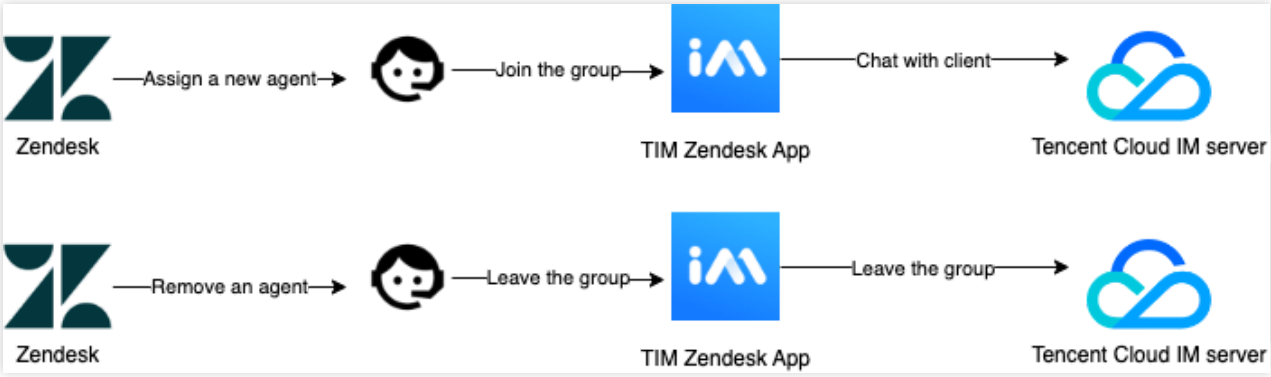
Otherwise submit a new ticket to Zendesk.

Login to the server and use the returned ticket.id to create a group.

The process of Tencent Cloud IM for Zendesk is:



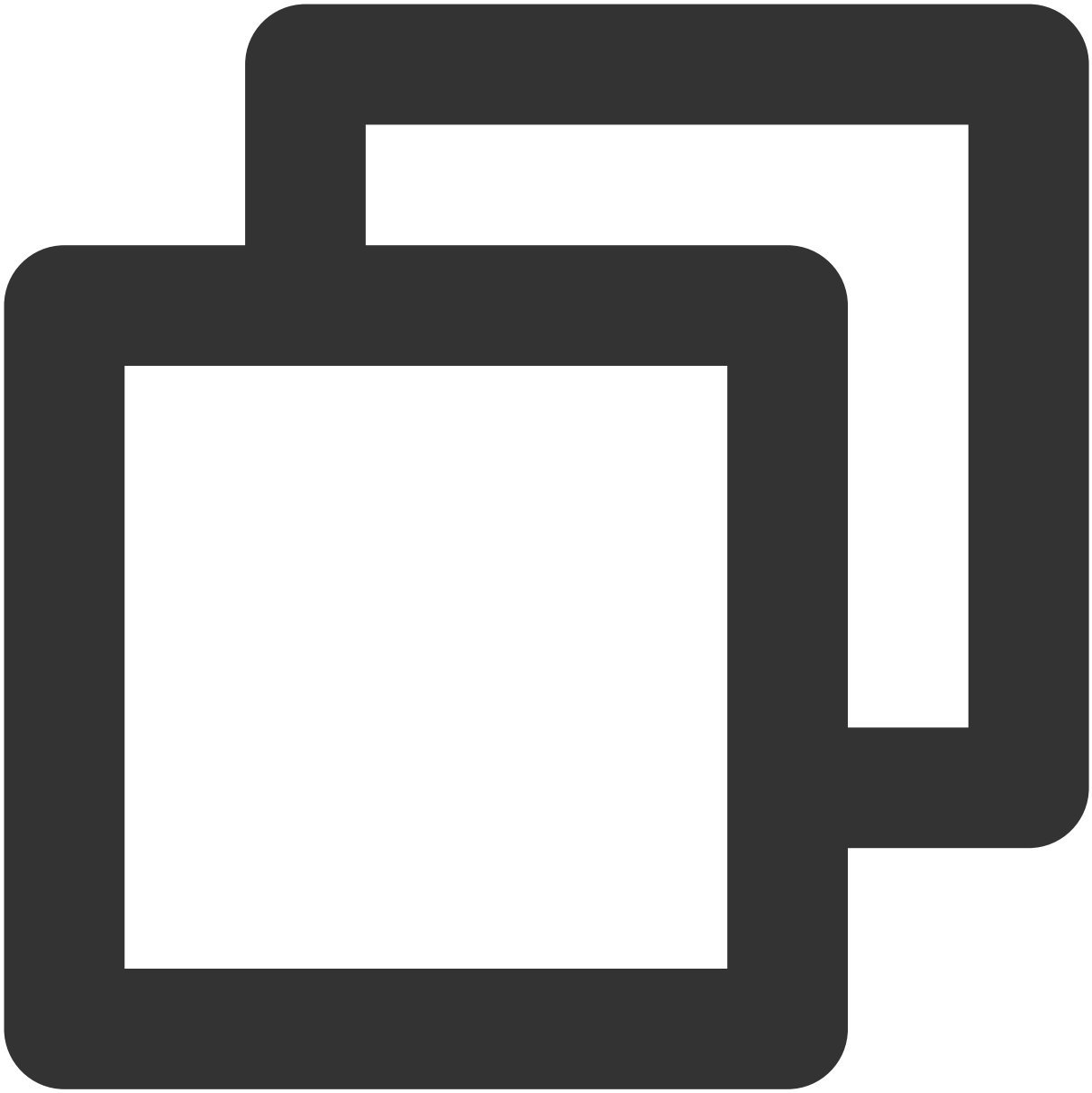
The process of updating the ticket:



Client Side Construction

You'll need to develop a frontend project to accomplish the chatting features, and a backend project to invoke Zendesk requests. Don't be panic about the works need to do, the frontend can be done smoothly with TUIKit like piling up blocks. TUIKit is a set of TUI components based on IM SDKs, which provides independent components including conversation, chat, searching, relationship chain, group, and audio/video call. TUIKit currently covers platforms including iOS, Android, Web and Flutter, and we're working on React Native as well. Stay tuned on the channel to get the lastest status. By applying TUIKit, you just have to define your strategy to invoke the `createTicket` API.

Here's all the code to build the backend project:



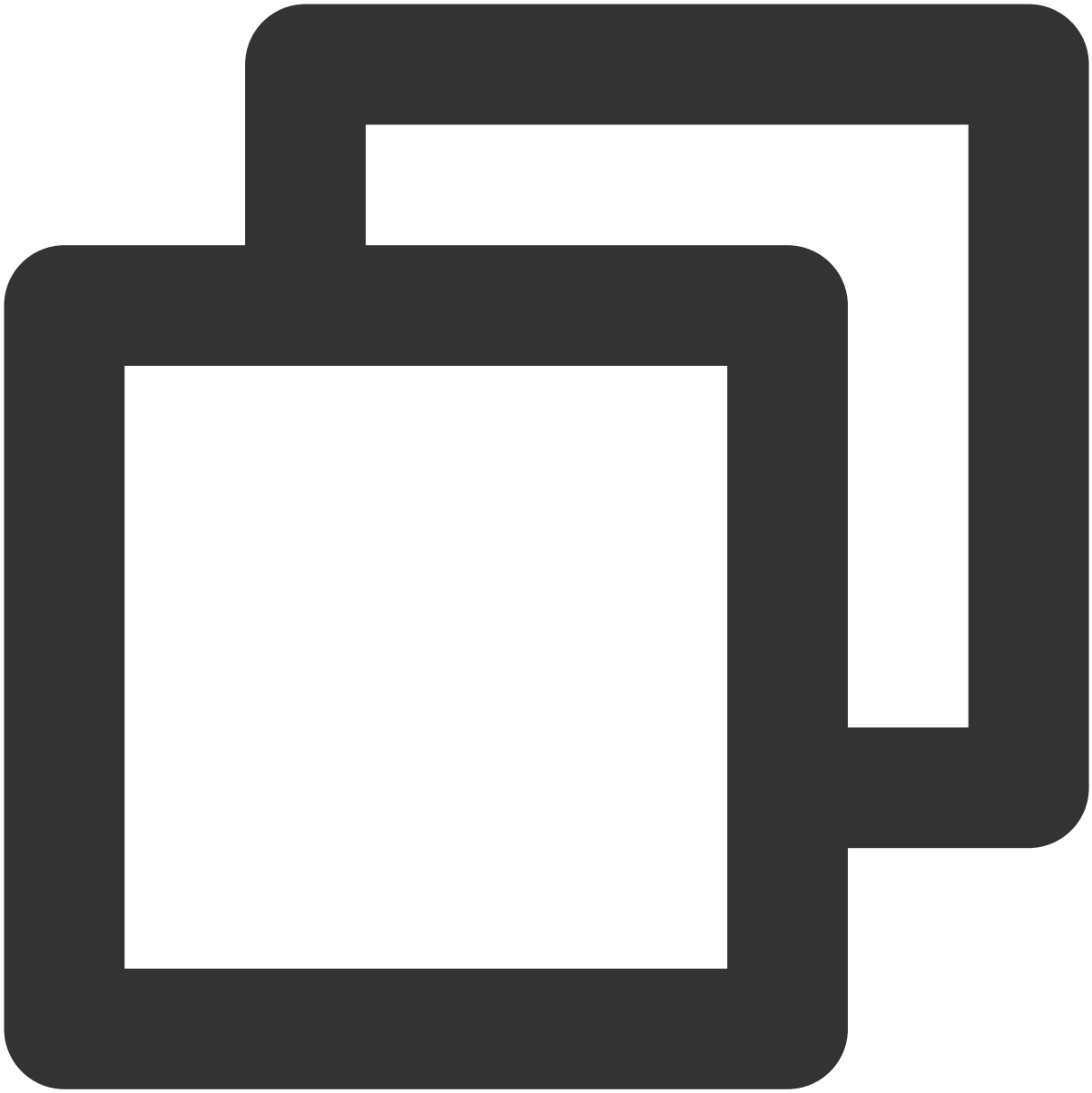
```
const express = require('express');
const axios = require('axios').default;
var TLSSigAPIv2 = require('tls-sig-api-v2'); // Generate UserSig for TIM
require('dotenv').config();

const app = express();
app.use(express.json());

const port = process.env.PORT || 15000;
const YOUR_SDKAPPID = process.env.YOUR_SDKAPPID || 0;
const YOUR_SECRET = process.env.YOUR_SECRET || '';
```

```
const ADMIN_USERID = process.env.ADMIN_USERID || '';

const zendeskAxioInstance = axios.create({
  baseURL: `https://${process.env.SUB_DOMAIN}.zendesk.com/api/v2/`,
  auth: {
    username: process.env.EMAIL || '',
    password: process.env.TOKEN || ''
  },
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  timeout: 35000
});

app.use(express.json());

app.use(function (req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', '*');
  next();
});

app.post('/createticket', async (req, res) => {
  const { userId, caseInfo } = req.body;
  if (!userId) return res.status(500).send('Missing userId');

  const zendeskCase = await createTicket(caseInfo);
  if (!zendeskCase.success) return res.status(500).send('Case creation failed');

  const groupId = zendeskCase.id;
  const result = await createGroup(
    groupId,
    userId,
    caseInfo.subject || groupId
  );
  if (result.ErrorCode !== 0)
    return res.status(500).send('Group creation failed');
  sendGreeting(result.GroupId);

  res.status(200).send(result);
});

const createTicket = async function (caseInfo) {
  const { subject, desc, name, email } = caseInfo;
  const data = {
    request: {
      requester: {
```

```
        name: name || 'Anonymous customer',
        email: email
      },
      subject: subject,
      comment: {
        body: desc
      }
    }
  };
  try {
    const result = await zendeskAxioInstance({
      method: 'POST',
      url: '/requests.json',
      data: data
    });
    return {
      id: result.data.request.id,
      success: true
    };
  } catch (e) {
    return { id: undefined, success: false, error: e };
  }
};

const generateUserSig = function () {
  const expires = 600;
  const api = new TLSSigAPIv2.Api(YOUR_SDKAPPID, YOUR_SECRET);
  return api.genSig(ADMIN_USERID, expires);
};

const generateRandom = function () {
  return Math.floor(Math.random() * 4294967295);
};

const createGroup = async function (groupId, userId, groupName) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    Type: 'Public',
    Name: groupName,
    GroupId: `ZENDESK#${groupId}`,
    ApplyJoinOption: 'FreeAccess',
    MemberList: [{ Member_Account: userId }]
  };
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
    const groupRes = await axios.post(url, data);
```

```
    return groupRes.data;
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e };
  }
};

const sendGreeting = async function (groupId) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    GroupId: groupId,
    Content: "We're assigning an agent to your subject, please wait..."
  };
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/send_group_system_
  try {
    const groupRes = await axios.post(url, data);
    return groupRes.data;
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e };
  }
};

app.listen(process.env.PORT || port, () =>
  console.log(`Example app listening on port ${port}!`)
);
```

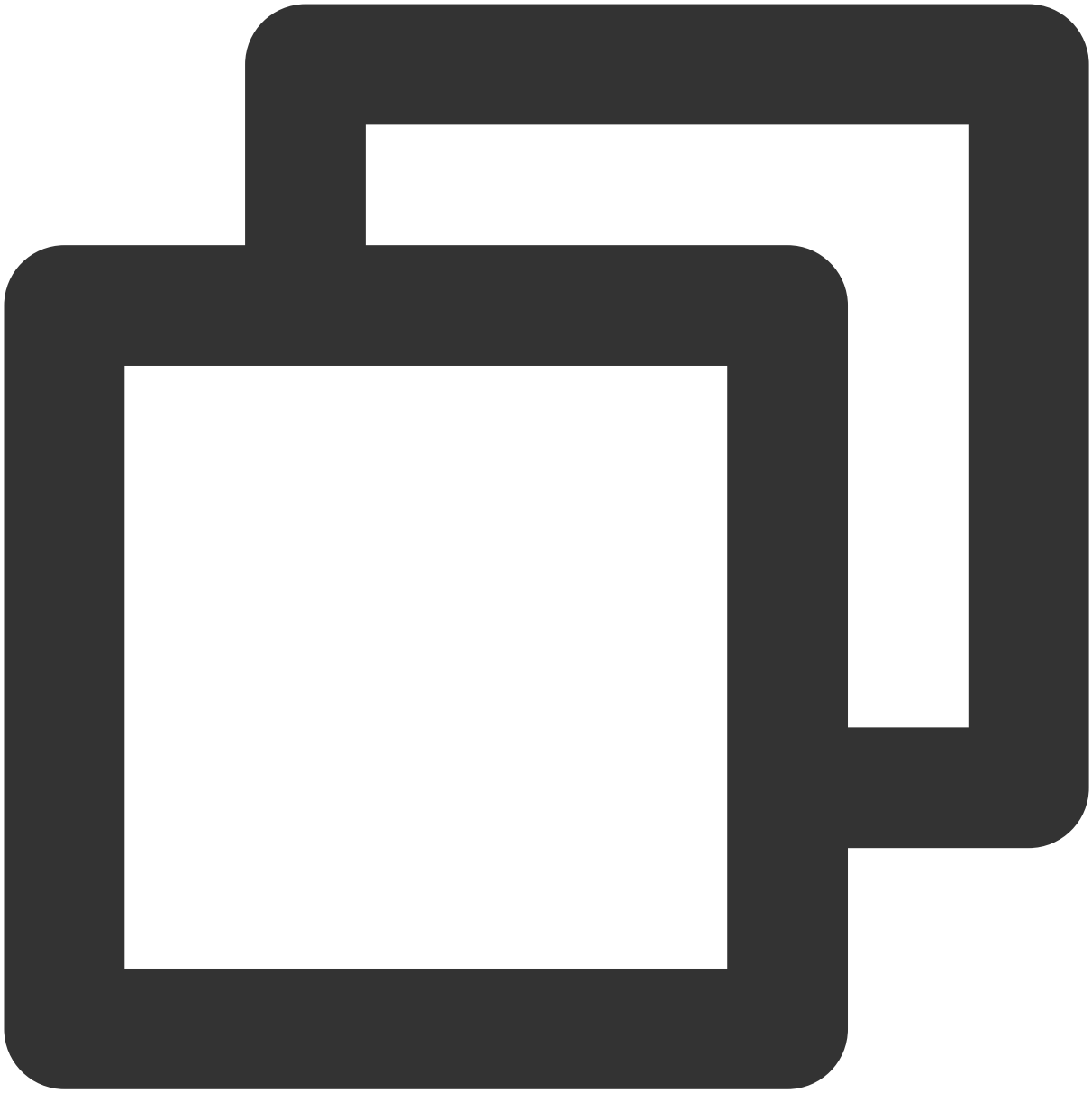
Let's break down the code.



```
const port = process.env.PORT || 15000;
const YOUR_SDKAPPID = process.env.YOUR_SDKAPPID || 0;
const YOUR_SECRET = process.env.YOUR_SECRET || '';
const ADMIN_USERID = process.env.ADMIN_USERID || '';
```

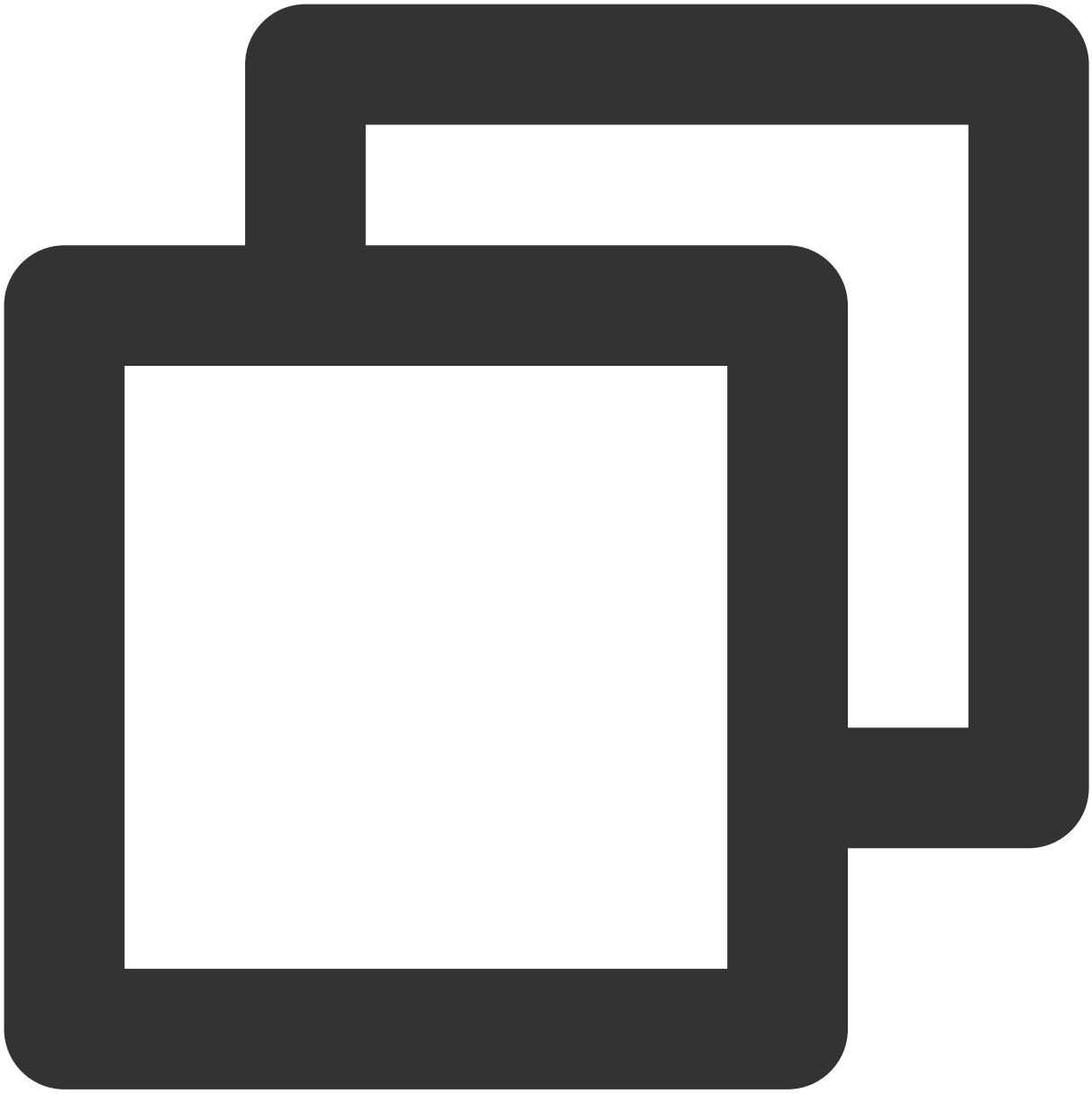
The first part requires you to set your Tencent Cloud IM identification information as the environment variables. You can find all the information here.

`Caveat: Do not note down your Secret in your code for it might be leaked to the Internet.`



```
const zendeskAxioInstance = axios.create({
  baseURL: `https://${process.env.SUB_DOMAIN}.zendesk.com/api/v2/`,
  auth: {
    username: process.env.EMAIL || '',
    password: process.env.TOKEN || ''
  },
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  timeout: 35000
});
```

Secondly, config Zendesk request.



```
// Create zendesk ticket
const createTicket = async function (caseInfo) {
  const { subject, desc, name, email } = caseInfo;
  const data = {
    request: {
      requester: {
        name: name || 'Anonymous customer',
        email: email
      },
```

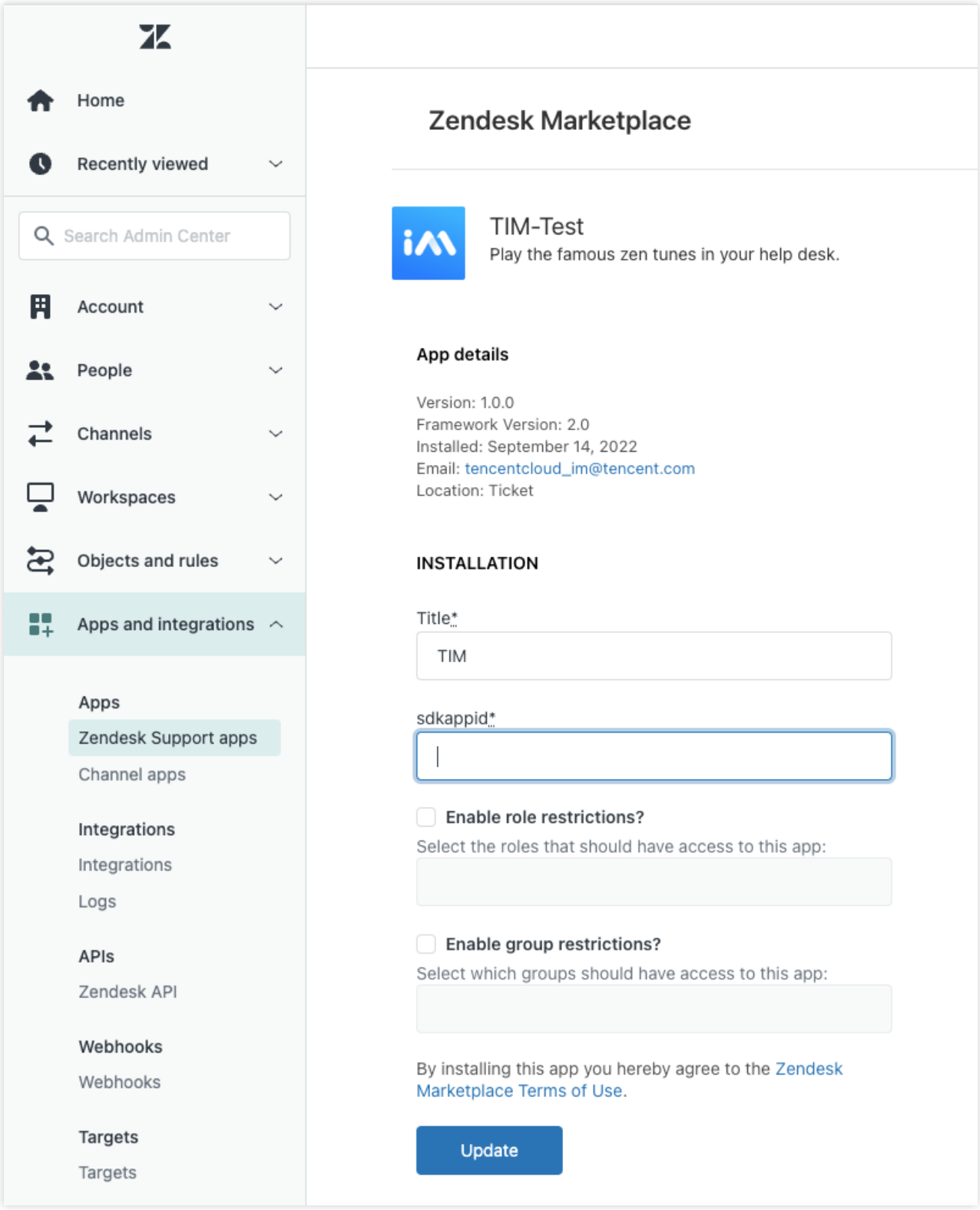
```
      subject: subject,
      comment: {
        body: desc
      }
    }
  };
  try {
    const result = await zendeskAxioInstance({
      method: 'POST',
      url: '/requests.json',
      data: data
    });
    return {
      id: result.data.request.id,
      success: true
    };
  } catch (e) {
    return { id: undefined, success: false, error: e };
  }
};

// Create IM group
const createGroup = async function (groupId, userId, groupName) {
  const sig = generateUserSig();
  const random = generateRandom();
  const data = {
    Type: 'Public',
    Name: groupName,
    GroupId: `ZENDESK#${groupId}`,
    ApplyJoinOption: 'FreeAccess',
    MemberList: [{ Member_Account: userId }]
  };
  const url = `https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkap
  try {
    const groupRes = await axios.post(url, data);
    return groupRes.data;
  } catch (e) {
    return { ErrorCode: -1, ErrorInfo: e };
  }
};
```

Last, create the Zendesk ticket by the input parameters and create an IM group with returned `zendeskCase.id` as the groupID, which uses `ZENDESK#` as the groupID prefix.

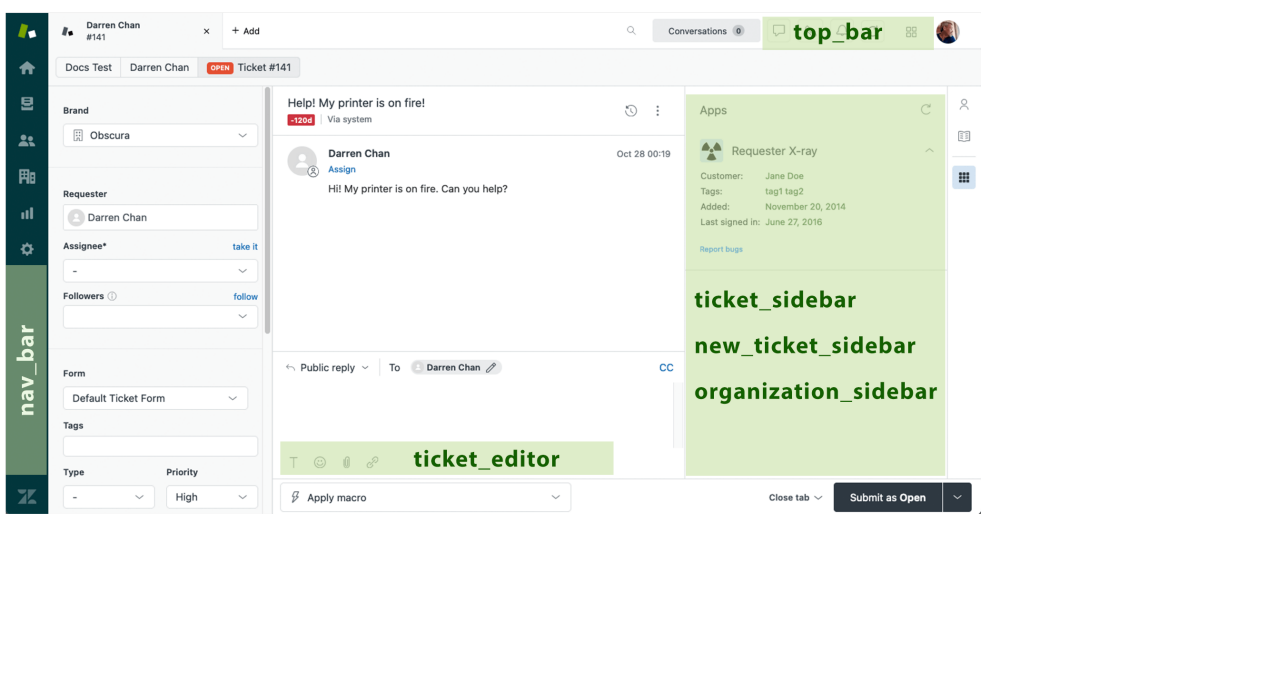
Test and publish the server code online for the client side.

TIM for Zendesk Installment

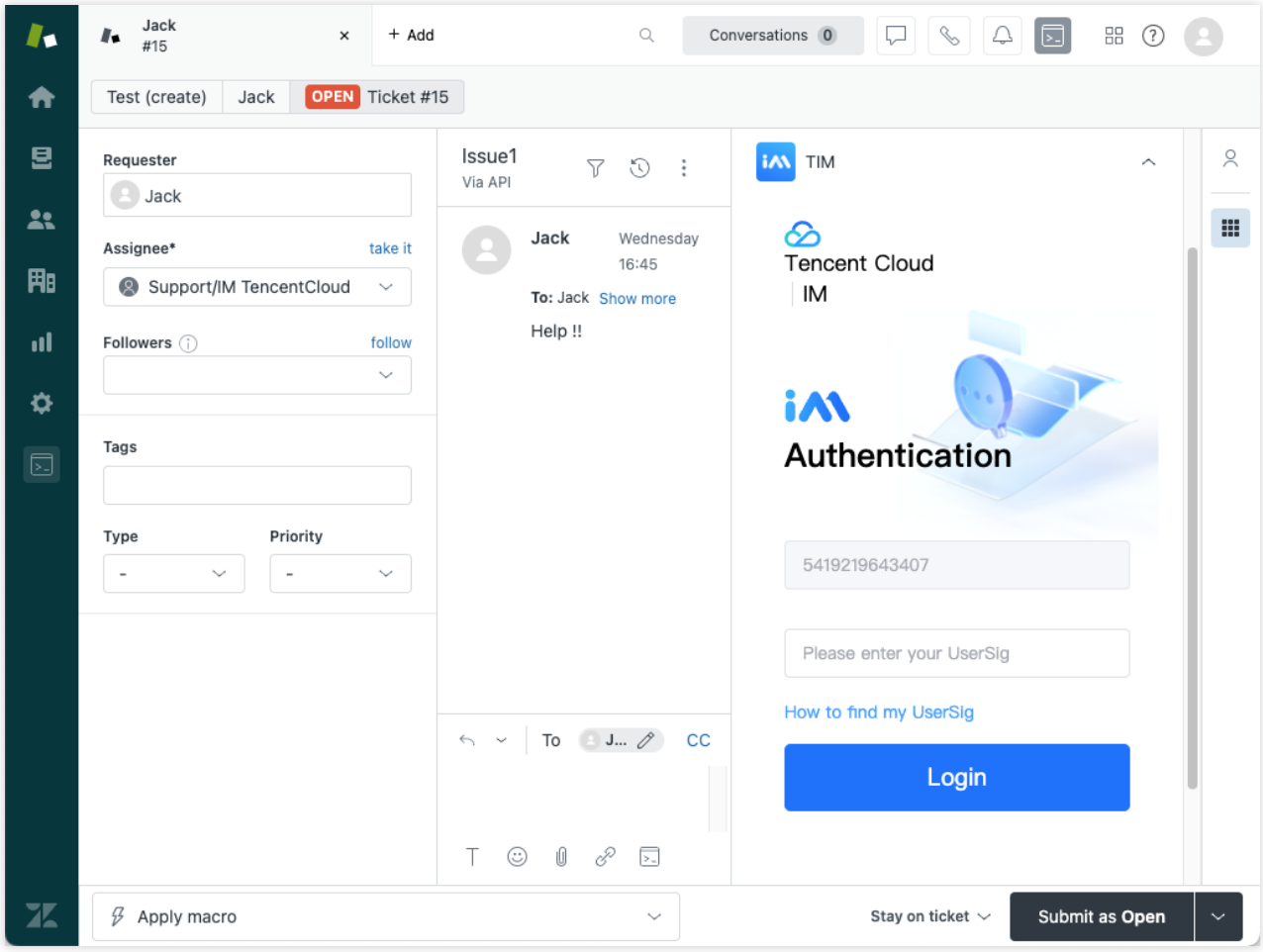
Install Tencent Cloud IM for Zendesk by searching Tencent Cloud IM for Zendesk on Zendesk marketplace.
Next, type your SDKAppId for the App to complete the installation.



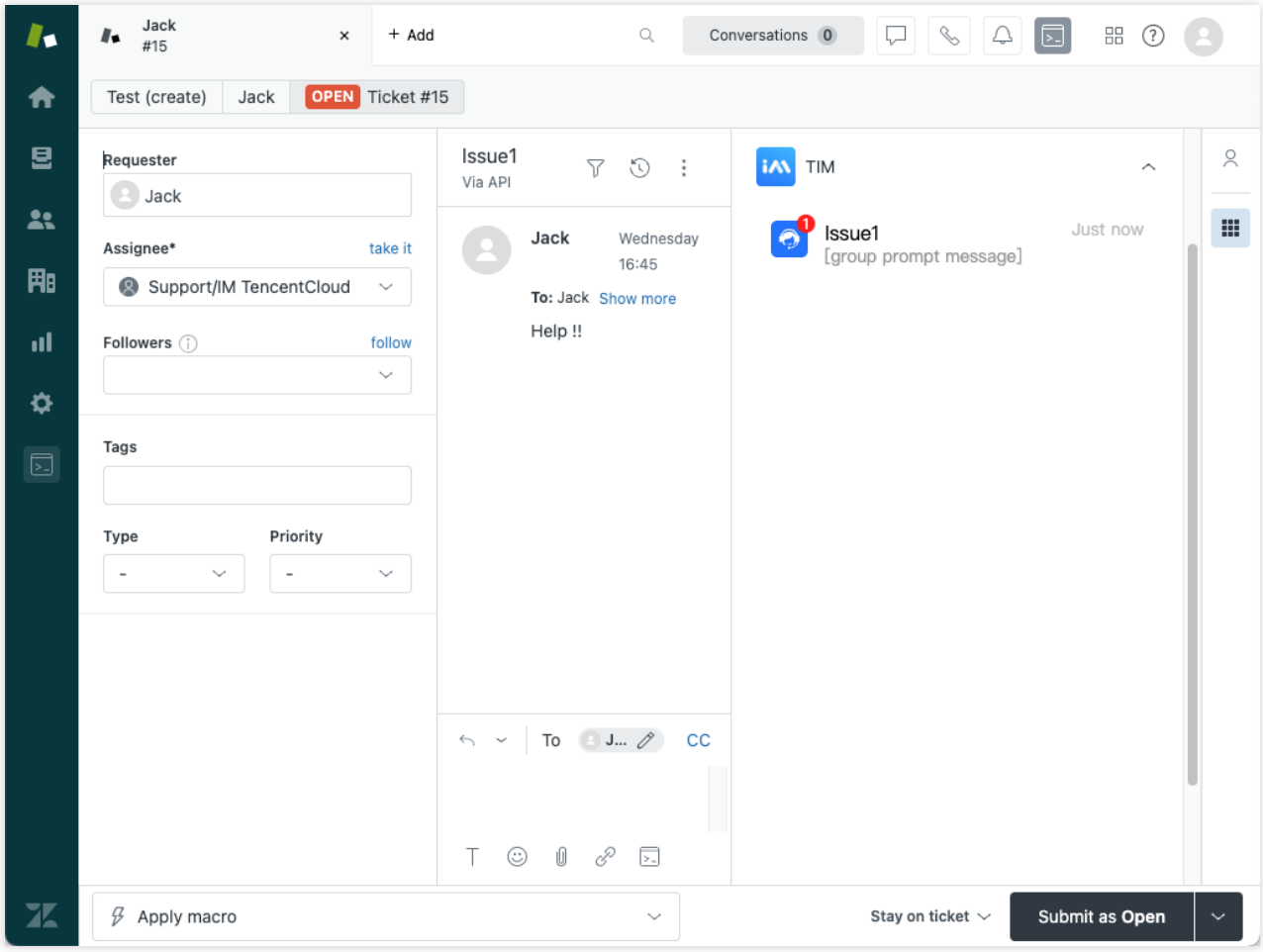
The App will show on the ticket bar area.

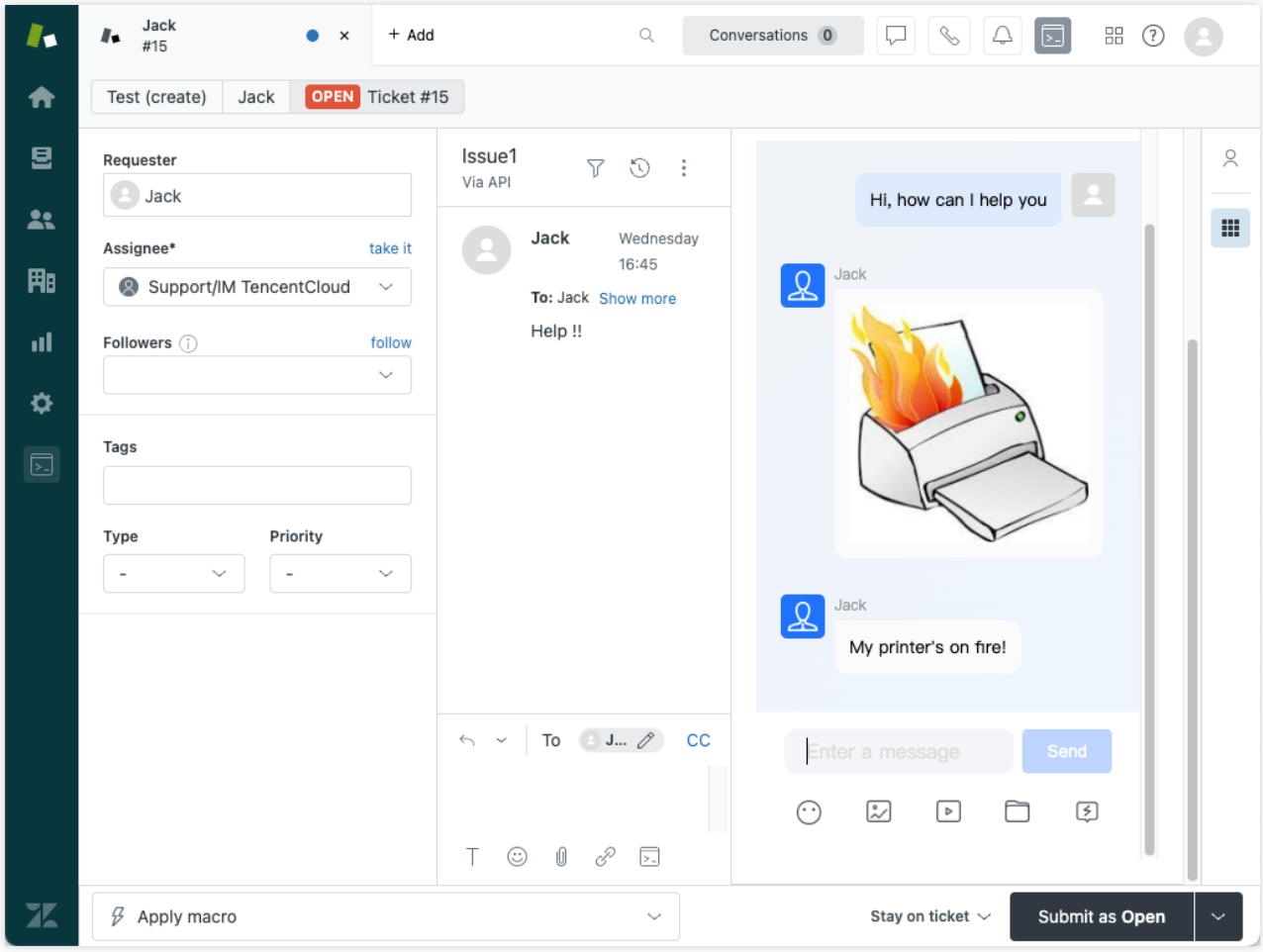


It uses agent's ID as UserID for Tencent Cloud IM, and you need to acquire UserSig for this UserID. See how to retrieve UserSig. Once logged in, the username in IM is automatically updated by the agent's name, which can be updated with other profile information on the profile page. Once logged in, you'll stay logged in until UserSig is expired or log out triggered.



And when an agent is assigned to the ticket, he/she will be invited to the IM group and chat with clients with various types of messages.





Also you can install the App privately, contact tencentcloud_im@tencent.com to get the lastest TIM package or the source code. Follow the instructions to upload private App here.

If the provided App is not what you expect, you may also develop your own private App. You may apply TUIKit to boost UI construction, and follow the instructions to create your own App. More Zendesk API References are listed here.

Conclusion

That's all you need to integrate Tencent Cloud IM with Zendesk. By now, you may have apprehended the process of the integration and the workflow of Zendesk and established your own client-agent-real-time communication App for Zendesk. If there's anything unclear or you have more thinkings about the integration with Zendesk, feel free to contact us!

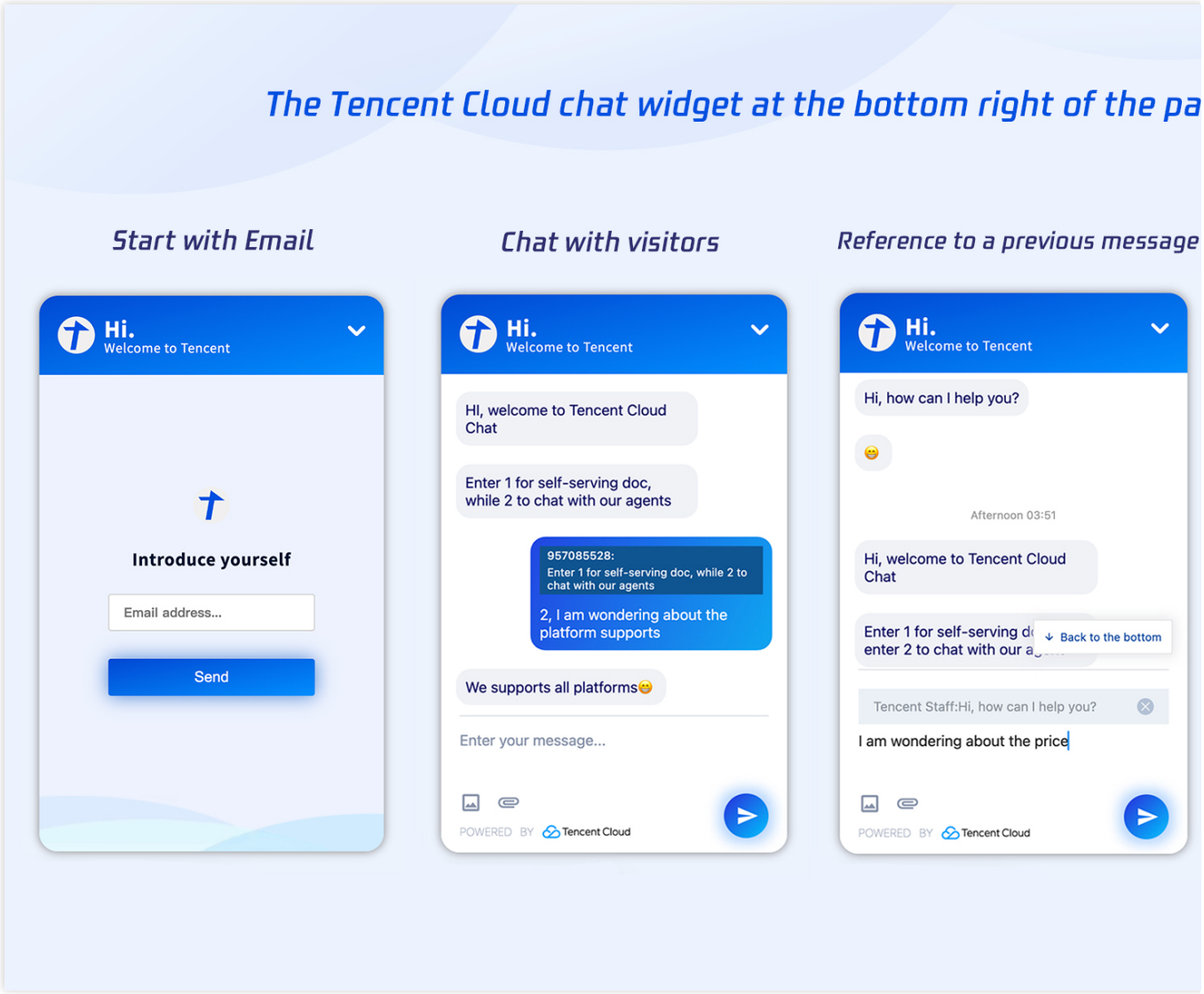
How to integrate chat widget to your Shopify online store

최종 업데이트 날짜: : 2024-02-07 17:30:52

Try it now with our Demo, password is `tencentim` .

Introduction

Shopify, the world's leading e-commercial SaaS platform, with numbers of merchants building their online shops based on it globally. Meanwhile, according to Gartner, Tencent Cloud Chat became the champion in the Chinese market and one of the most competent providers in Communication Platform as a Service (CPaaS) in the global market.
Naturally, it comes to us to bring you the solution of building the chat widget on your Shopify online store based on Tencent Cloud Chat, in order to convert more online shop visitors to your customers, and deliver personal customer support at any scale no matter where they are from.
After the integration, the effect is shown below. Also, you can experience it with our Demo, password is `tencentim` .



What you'll learn

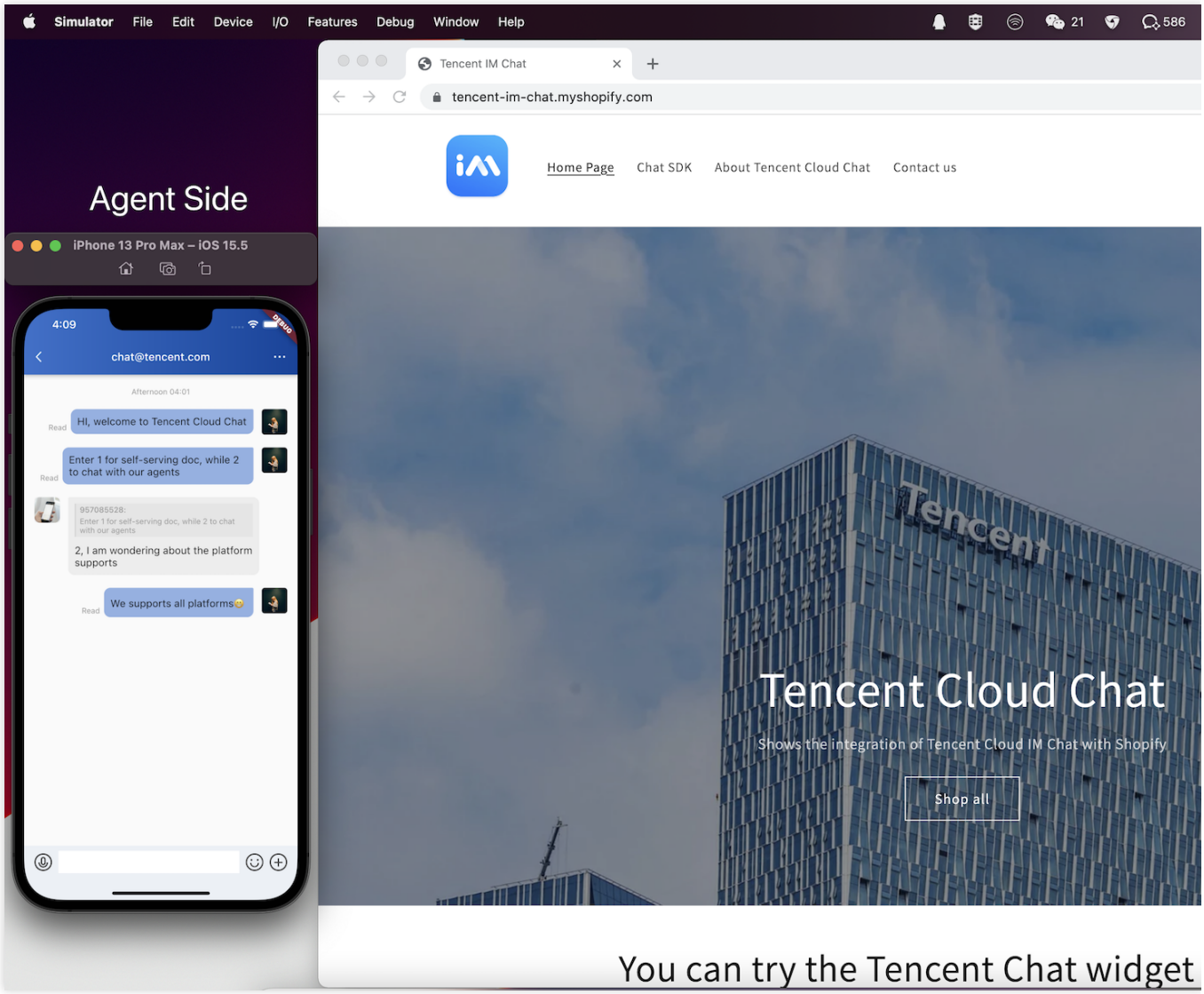
In this tutorial, we will discuss how to build this chat box on your Shopify online store with the codes we provided.

Initialize a Shopify app.

Building this app with the codes, related to the chat module, we provided.

Install this app to your Shopify store.

Enable it to communicate with your existing chat APP, for agent serving purposes.



Note:

For the agent side shows on the left of the picture above, you can implement it with our TUIKit or DEMO easily, or you can build your own APP with our Chat SDK.

Requirements

You've created a Tencent Cloud Chat APP, with the specific SdkAppID.

You've built a chat APP with the SDKs we provided, binding with this SdkAppID. It's recommended to build it based on our DEMO, if you do not have one.

You've created a Shopify Partner account and a development store.

You've installed Node.js 14.13.1 or higher.

You've installed a Node.js package manager: either npm, Yarn 1.x, or pnpm.

You've installed Git.

You're using the latest version of Chrome or Firefox.

You've installed an Online Store 2.0 theme, such as Dawn, that uses JSON templates.

You have at least Ruby 2.7.5 installed on your system.

Note:

The Shopify Partner account used in this tutorial is only for developing your own app purposes, you don't need to publicize this app to the Shopify App store.

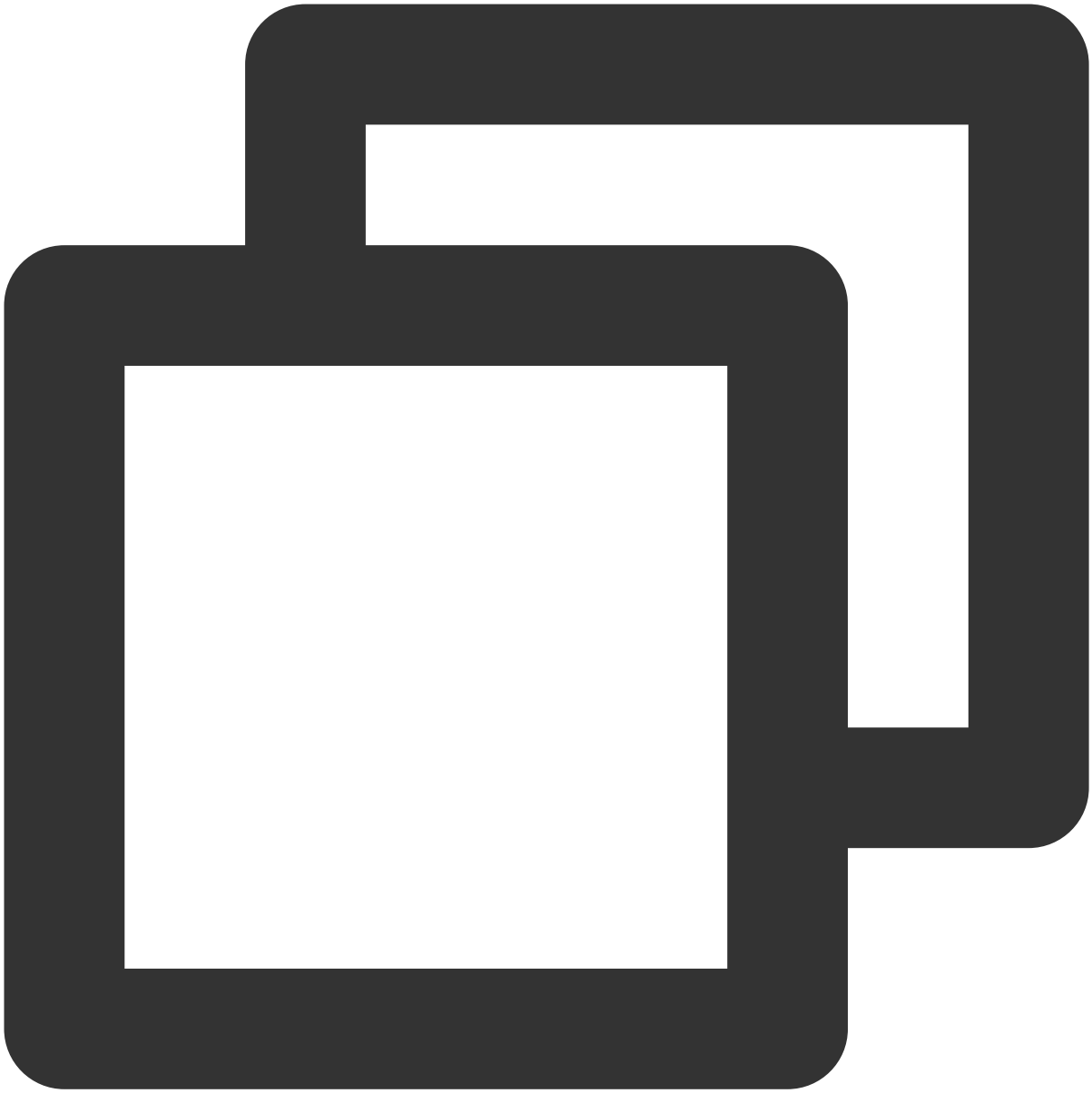
The development store is only for testing and previewing your app while developing, we will guide you to install your app to your online store by the end of this tutorial.

Let's get started

Step 1: Create a new Shopify app

You can create a new Shopify app using an npm, yarn, or pnpm command. Using npm as an example in this tutorial.

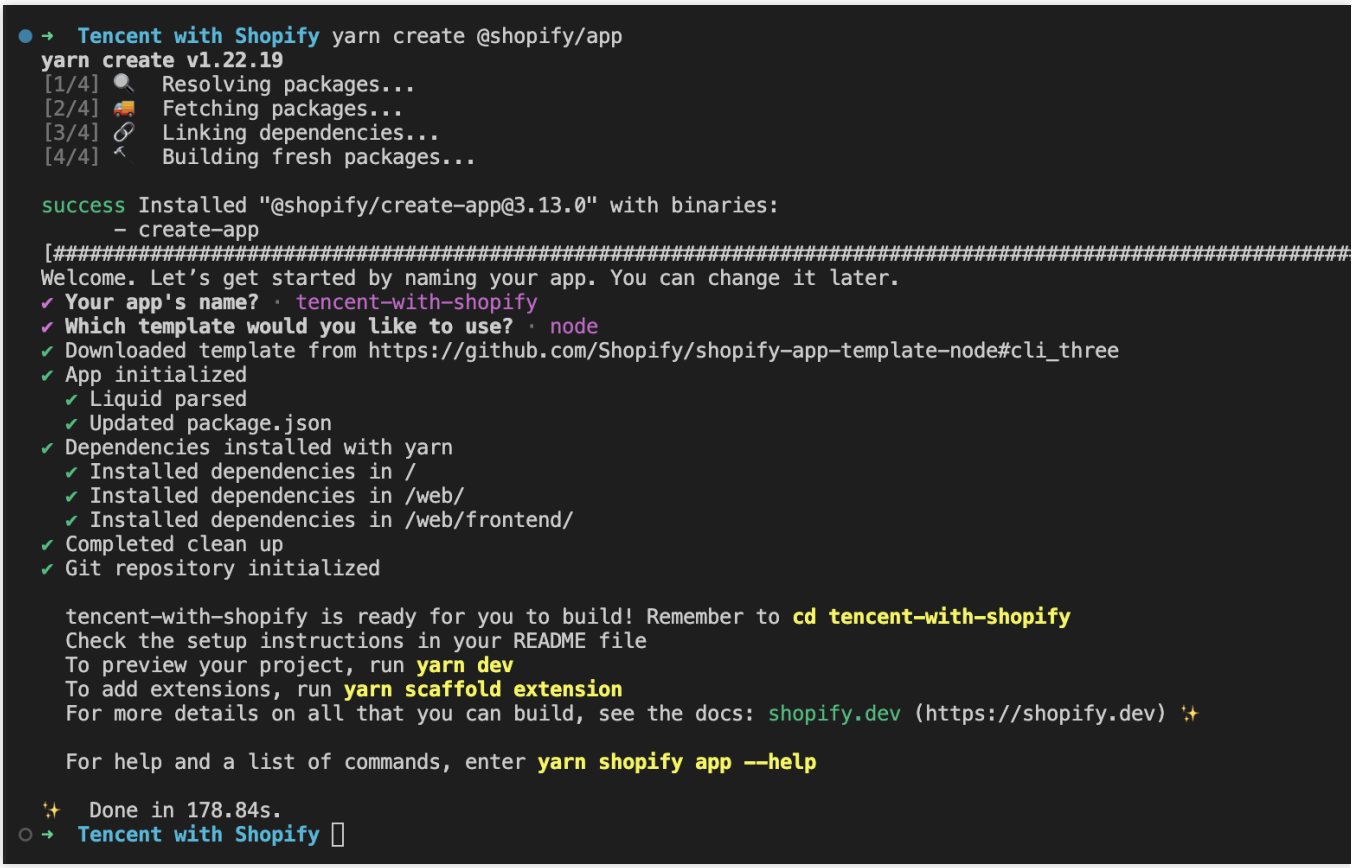
1. Navigate to the directory where you want to create your app. Your app will be created in a new subdirectory.
2. Run one of the following commands to create a new app, and select `node` as the template.



```
npm init @shopify/app@latest
```

A new app is created, and Shopify CLI is installed along with all of the dependencies that you need to build Shopify apps. Shopify CLI is also added as a dependency in your app's package.json.

The following image shows an app being successfully created in the terminal:

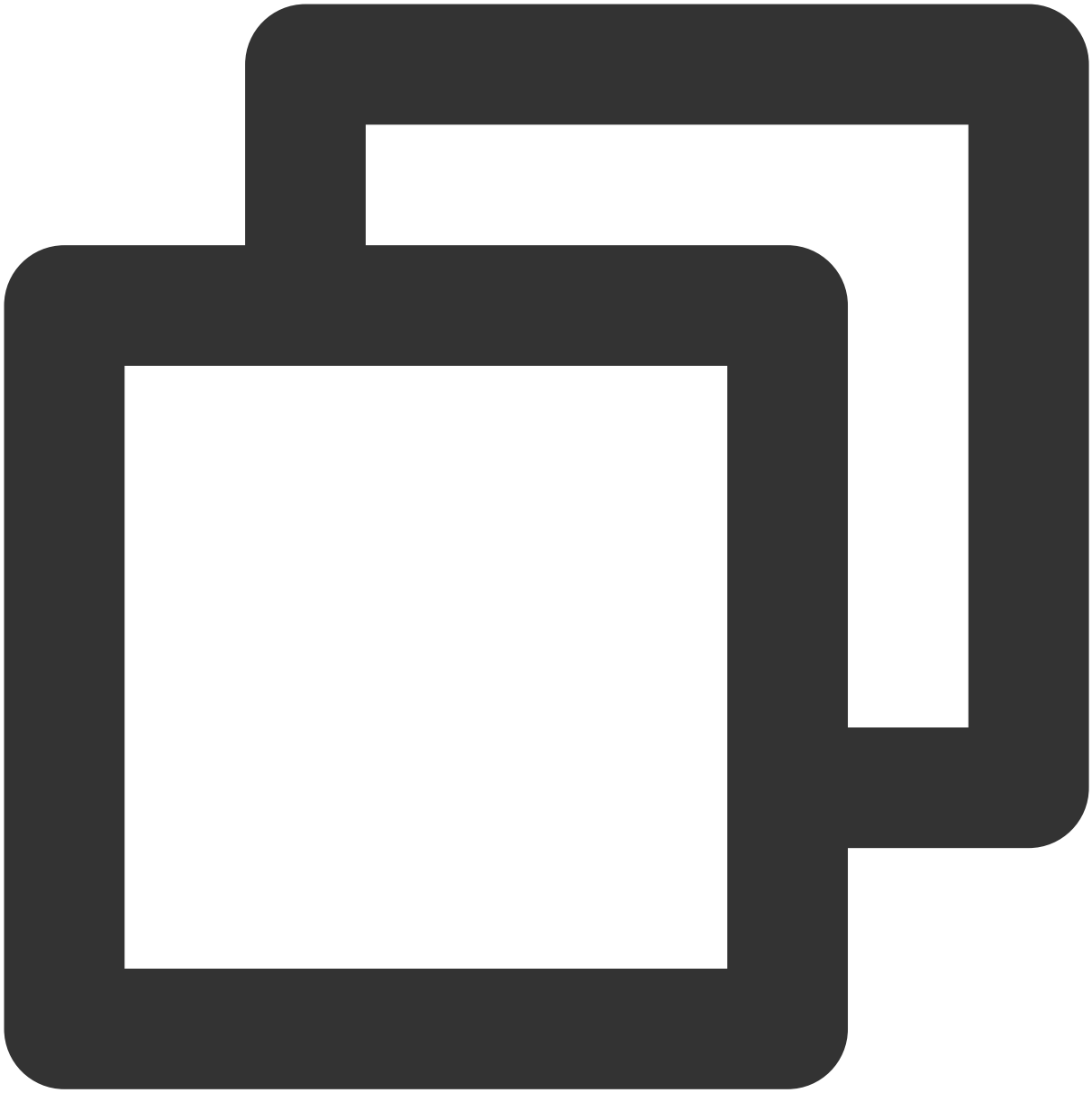


Step 2: Start a local development server

After your app is created, you can work with it by building the app and starting a local development server.

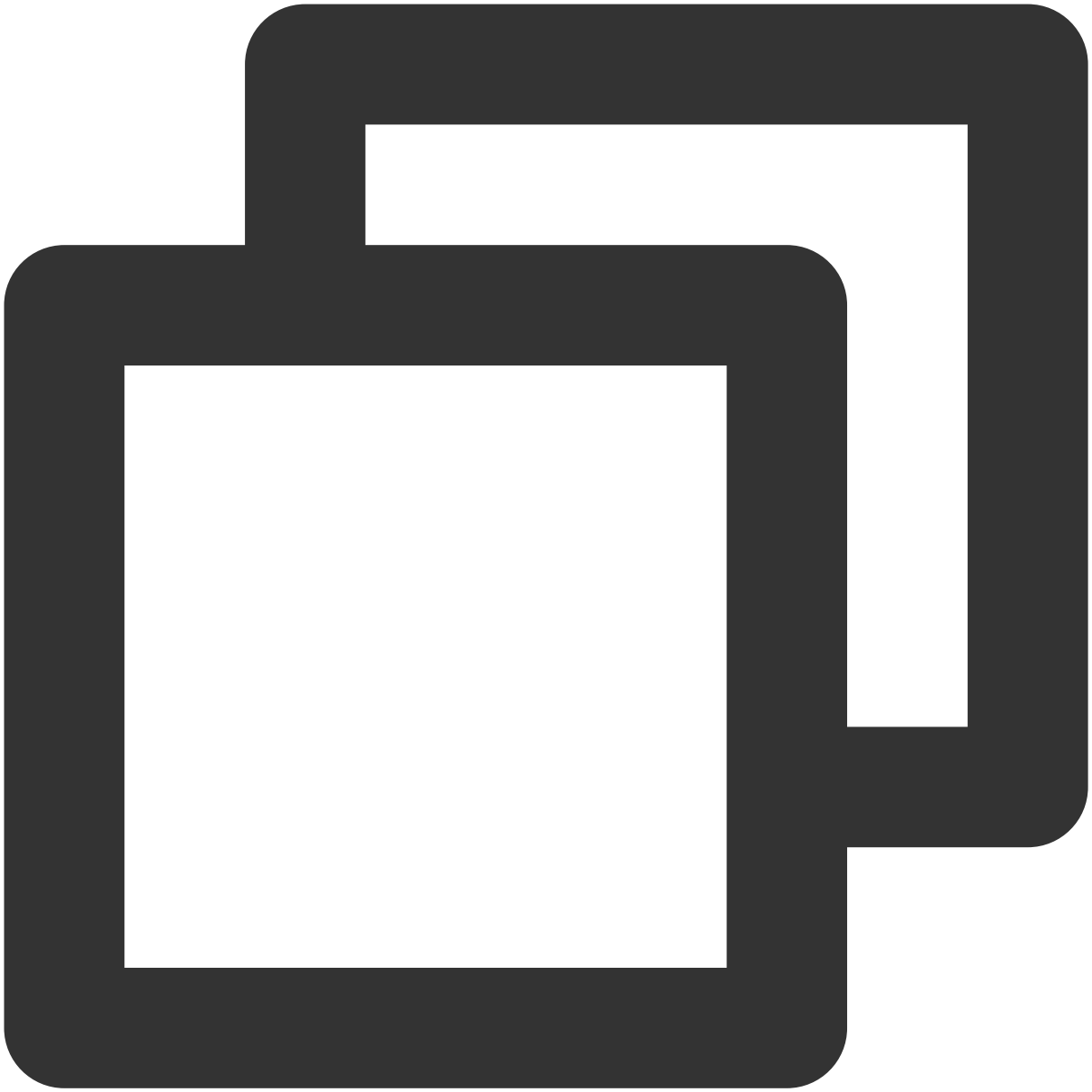
Shopify CLI uses ngrok to create a tunnel that allows your app to be accessed using a unique HTTPS URL. You need to create an ngrok account and auth token to preview your app using Shopify CLI.

1. Navigate to your newly-created app directory.



```
cd your-app
```

2. To start a local server for your app, run the following command:



```
npm run dev
```

Shopify CLI walks you through the following:

Logging into your Shopify Partner account and, if needed, selecting a Partner organization

Creating an app in the Partner Dashboard, and connecting your local app files to the app

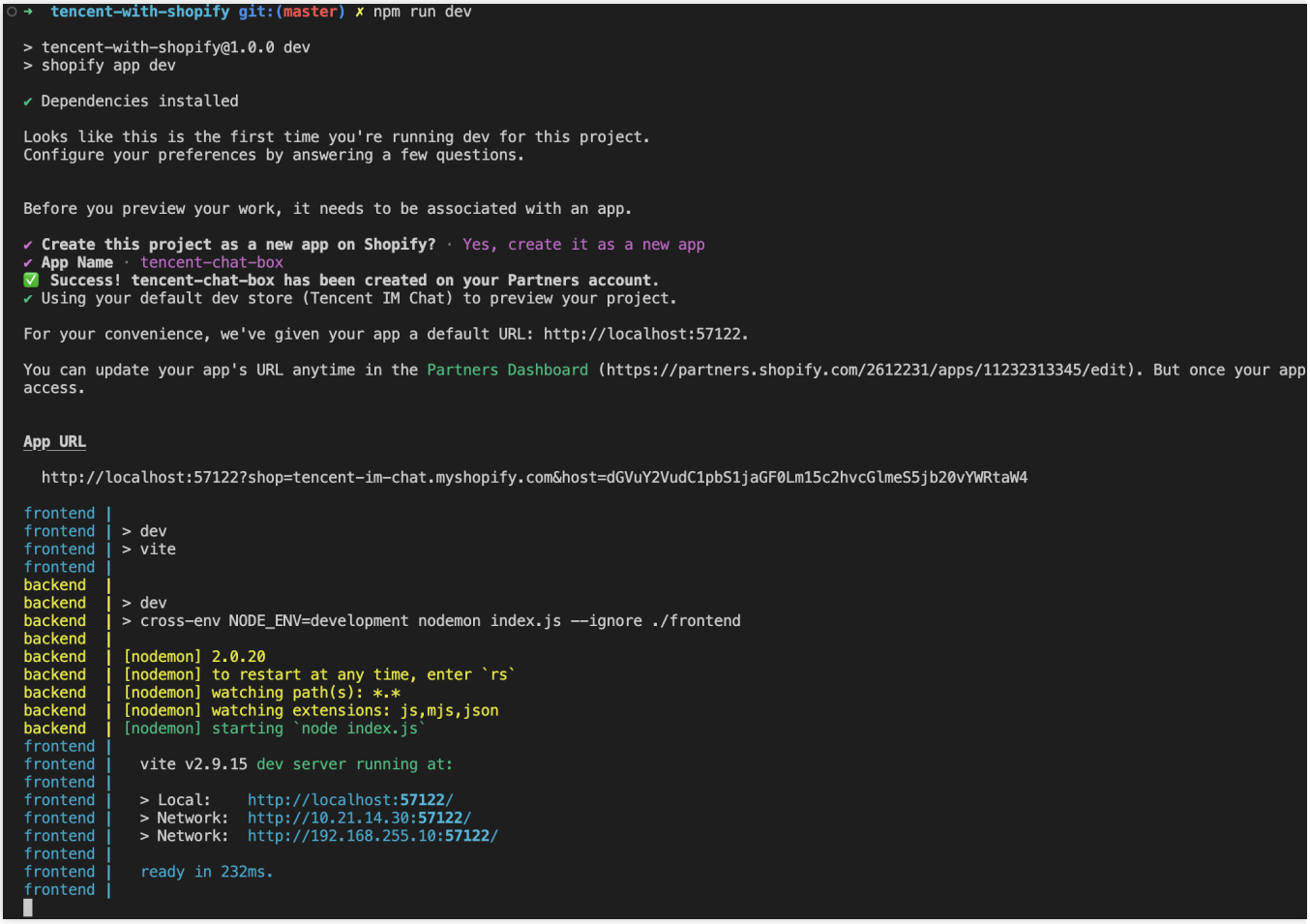
Storing your ngrok token, and then creating a tunnel between your local environment and the development store using ngrok

Note:

If you encounter ngrok errors in your browser when you try to preview your app or app extension, then consider using a tunnel created with your own tunneling software instead. Shopify recommends using Cloudflare Tunnel.

To use your own tunnel, pass your tunnel URL and port to the `dev` command with the --tunnel-url flag.

The following image shows a development server being started using `dev` :



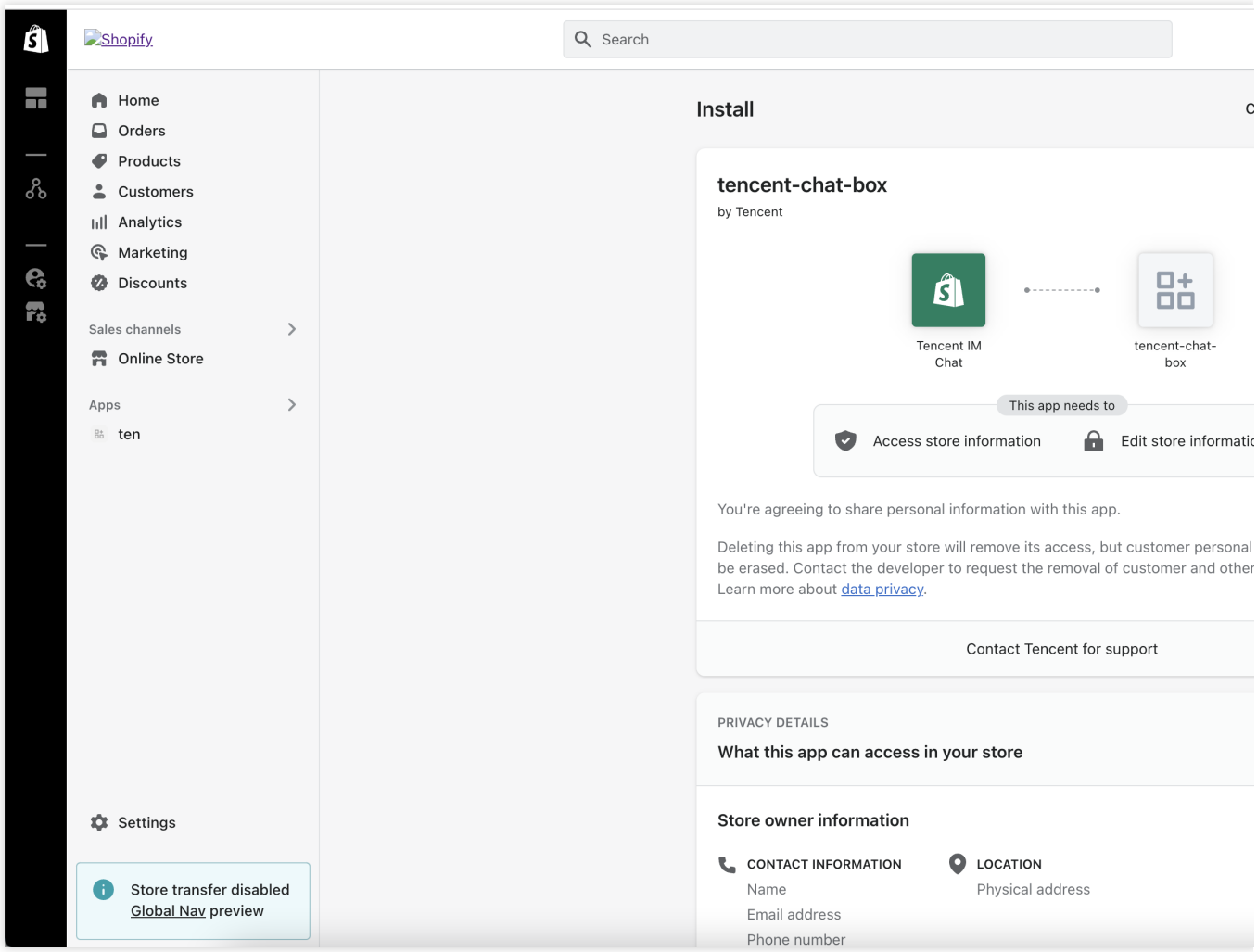
Step 3: Install your app on your development store

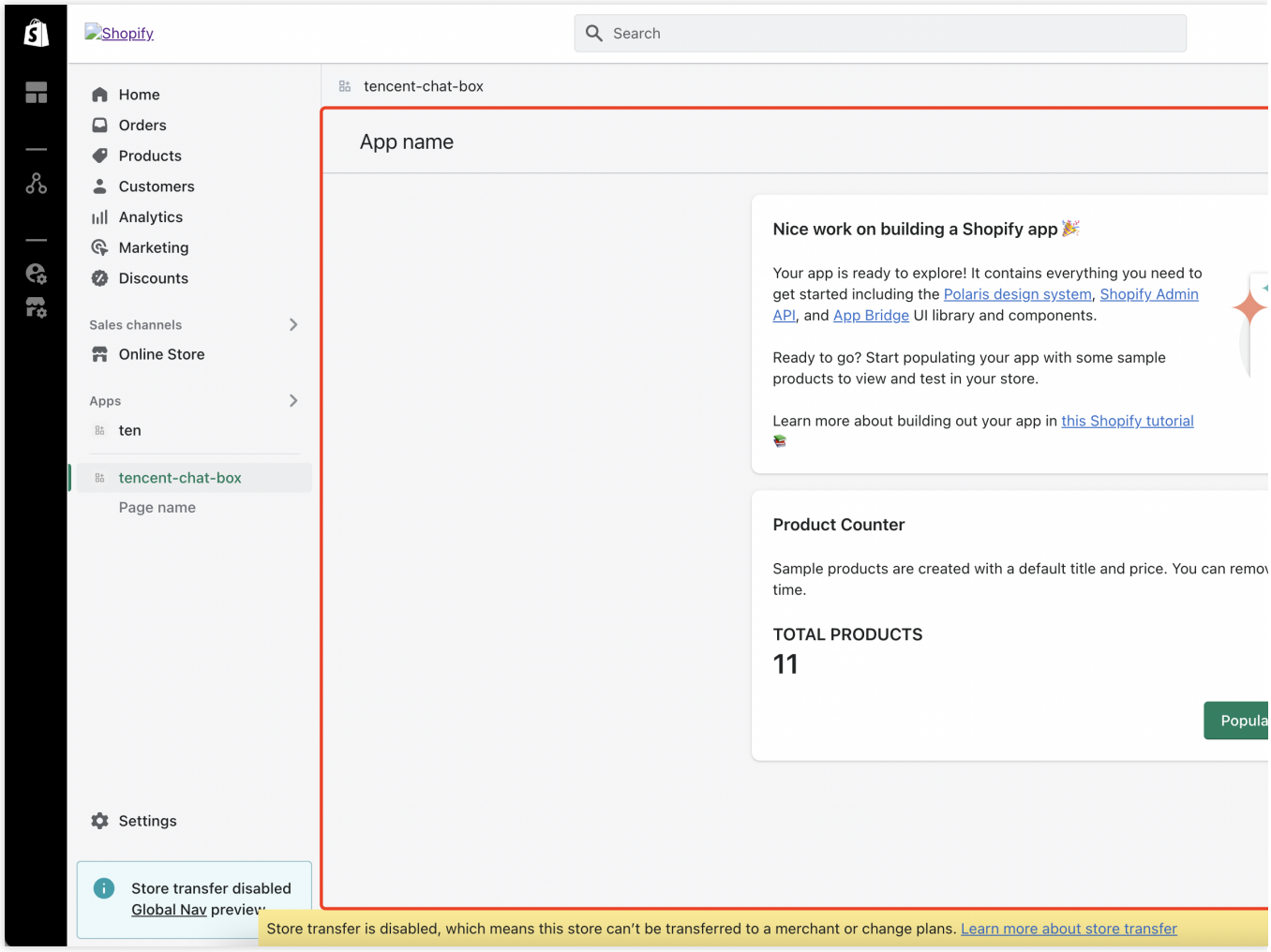
With the server running, open the URL in the App URL section of the terminal output in the previous step.

The URL follows the format `https://[tunnel_url]?shop=[dev_store].myshopify.com&host=[host]` , where `[host]` is the base64-encoded host parameter used by App Bridge, and represents the container the app is running in.

When you open the URL, you're prompted to install the app on your development store.

Click **Install app** to install the app in the store.





Step 4: Modify the frontend code in this admin dashboard

Note:

This step is optional, mainly to replace the default app page on admin with the configuration guidance page.

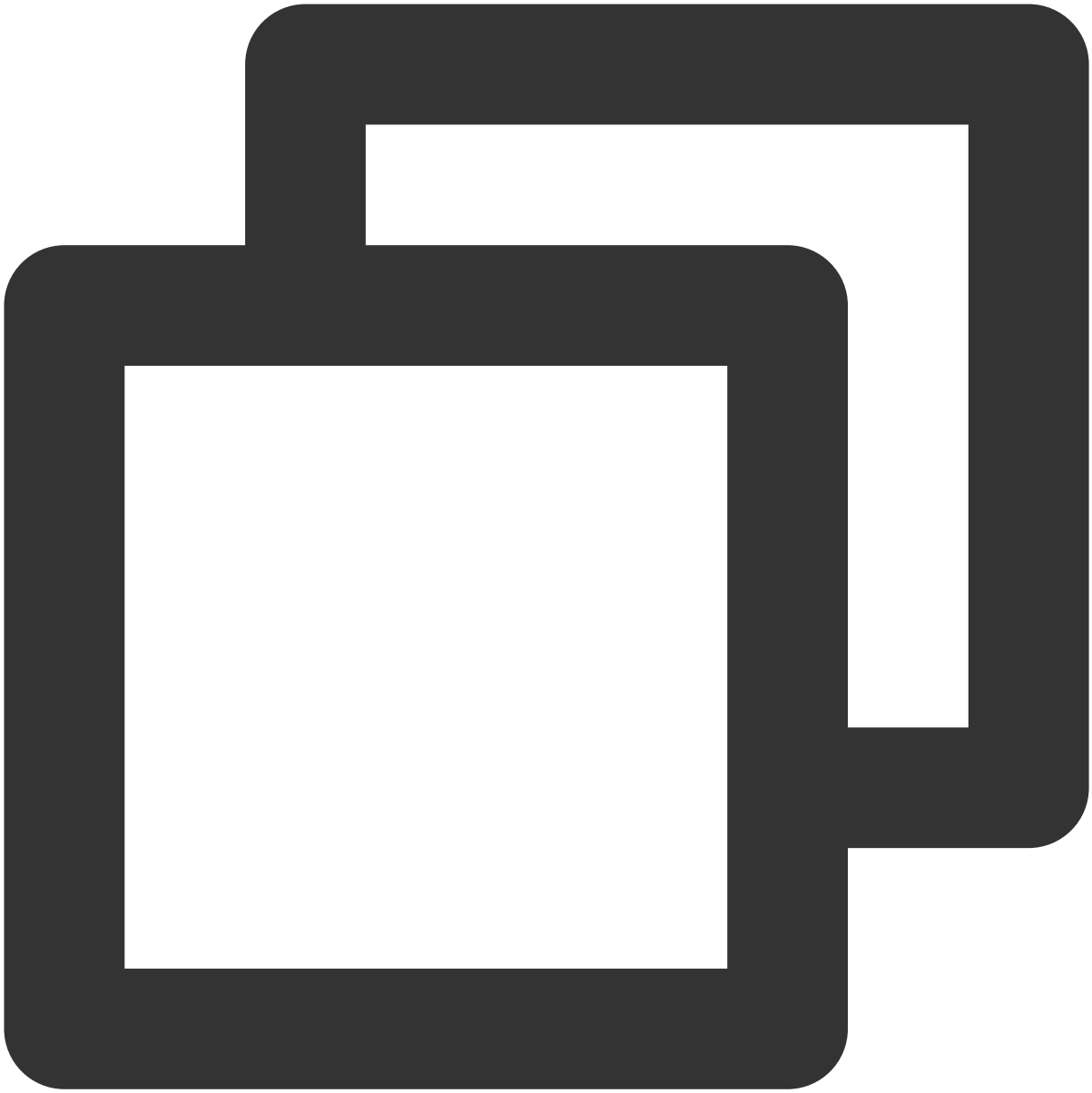
1. Replace the `web/frontend/pages/index.jsx` file.

Download the new file, named it as `index.jsx` and replace the file of `web/frontend/pages/index.jsx`.

2. Add the guide page.

Download the file, named it as `Guide.jsx`, and move it into `web/frontend/components/Guide.jsx`.

3. Add a line of export class to `web/frontend/components/index.js`.



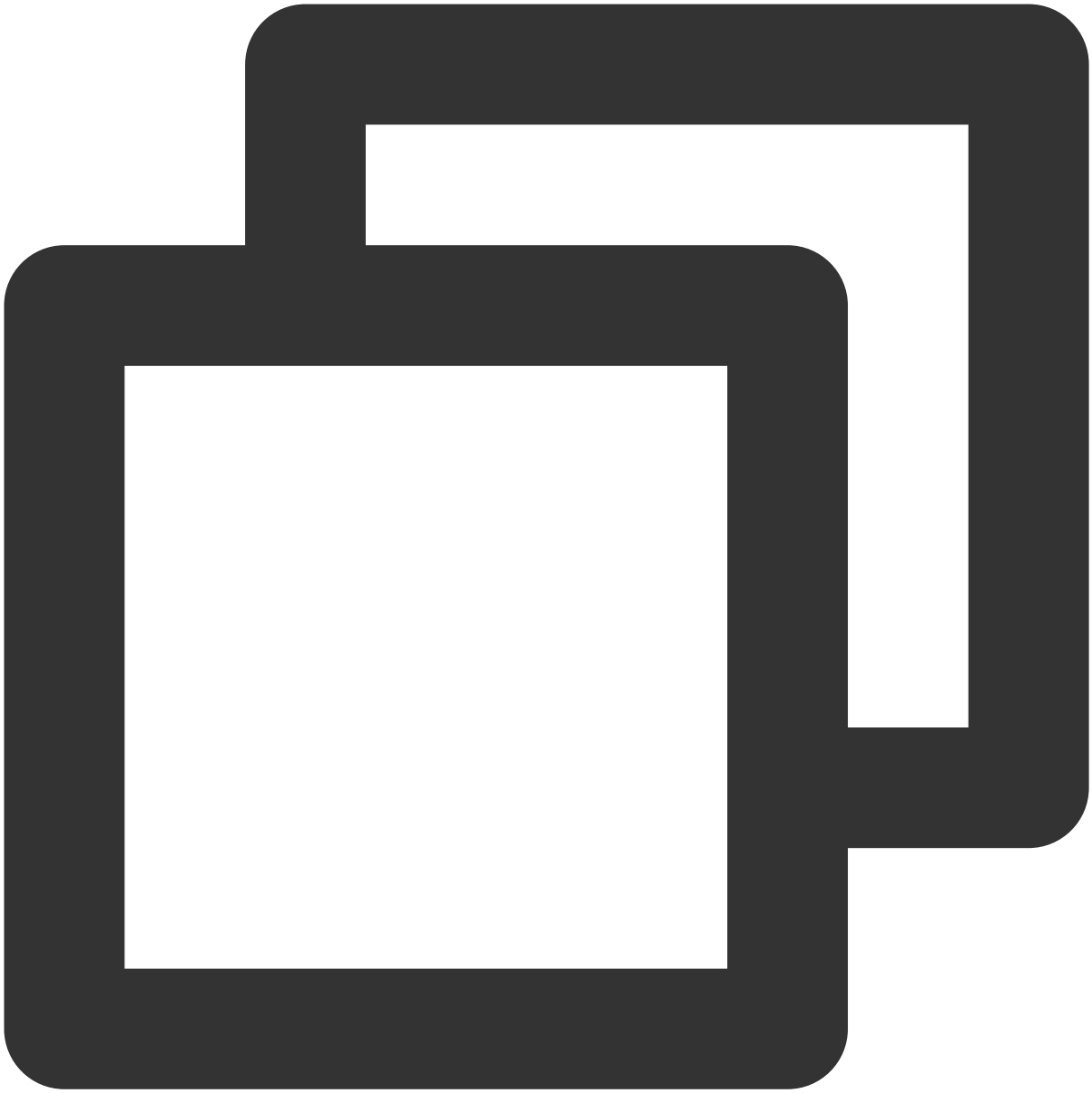
```
export { Guide } from "./Guide";
```

4. Add the file resources to `web/frontend/assets` .

Logo, named as `logo.png` .

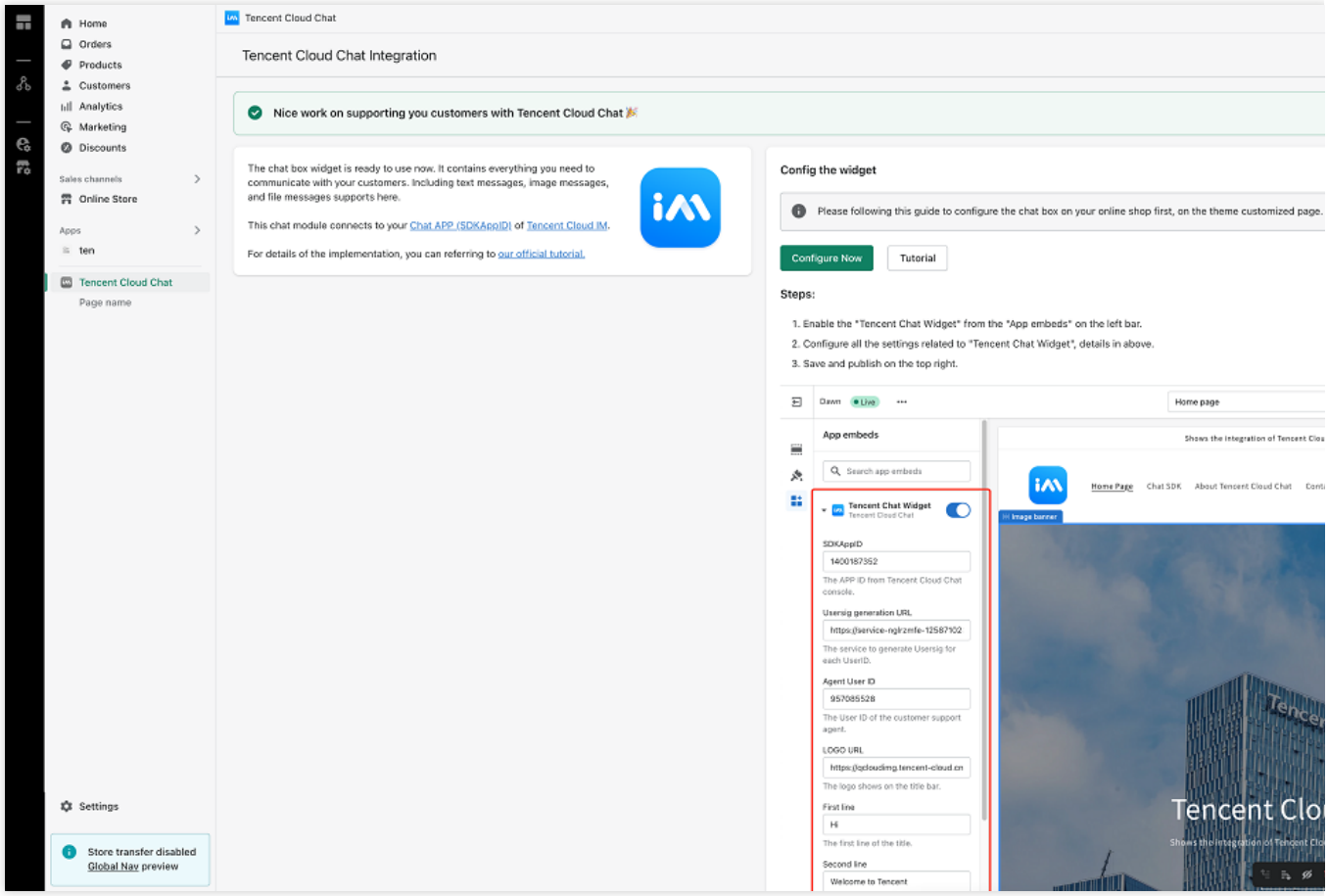
Demo instructions, named as `demo.png` .

Add the export to `web/frontend/assets/index.js` .



```
export { default as logo } from "./logo.png";
export { default as demoImage } from "./demo.png";
```

Now, the updating of the admin page is finished, you can refresh the page and see the following page.



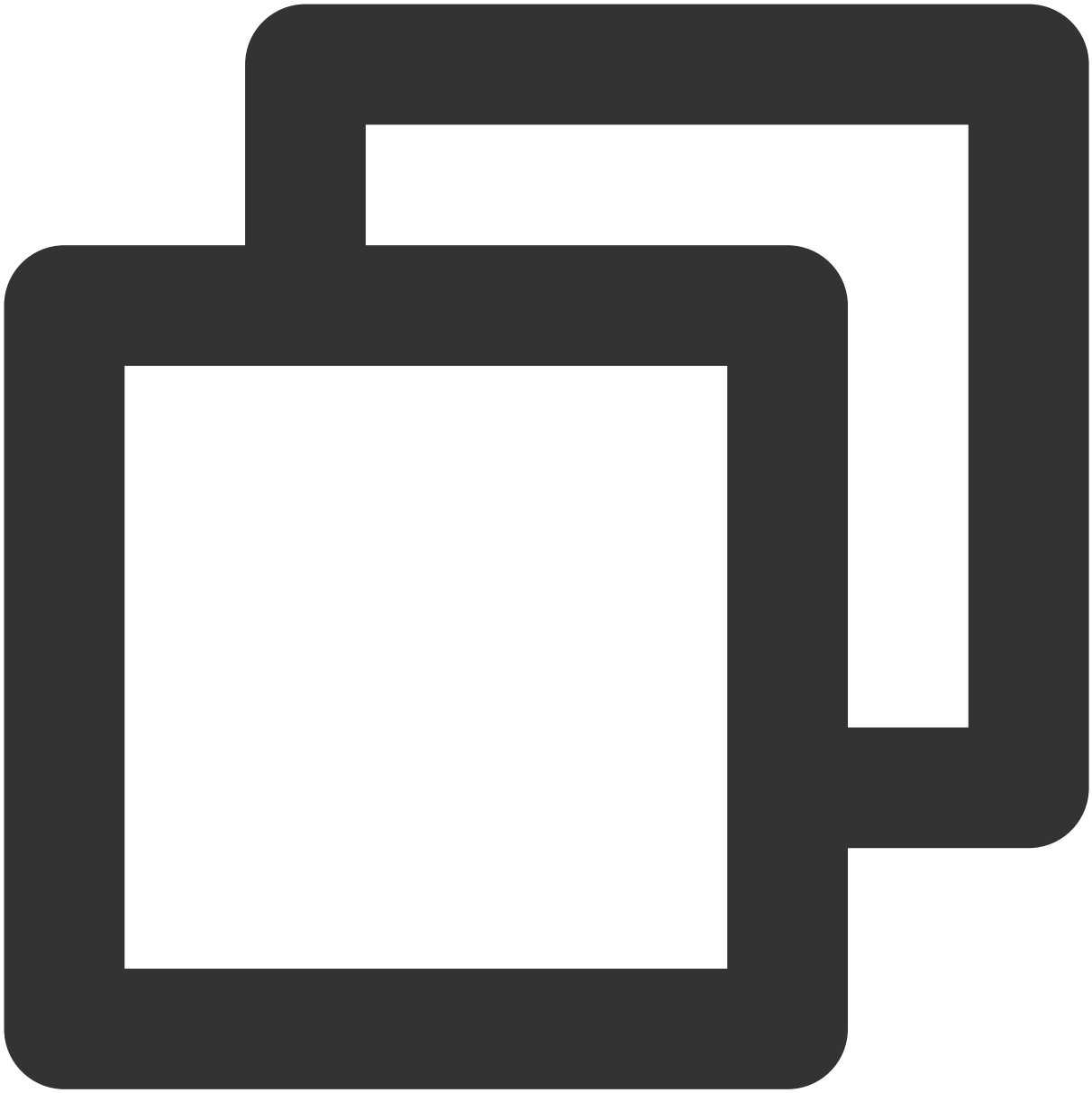
Step 5: Create a new extension

This extension will be the chat box widget shown at the bottom right of the online store.

You'll use Shopify CLI to generate a new extension first.

1. Navigate to the directory of the app that you want to add your extension to.

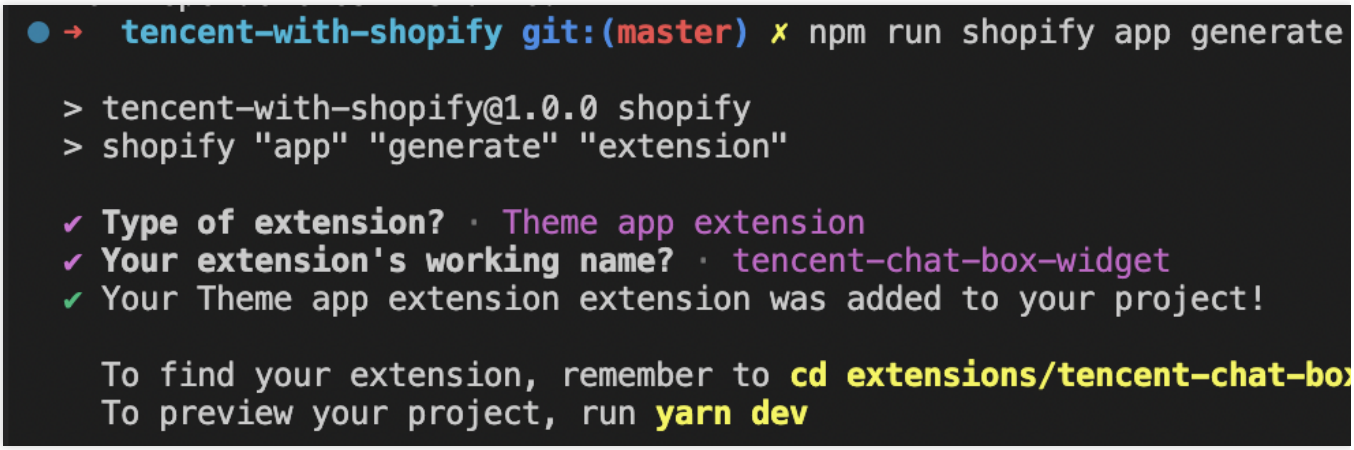
2. Run the following command to start creating the extension, choose `Theme app extension` as the `Type of extension` .



```
npm run shopify app generate extension
```

Note:

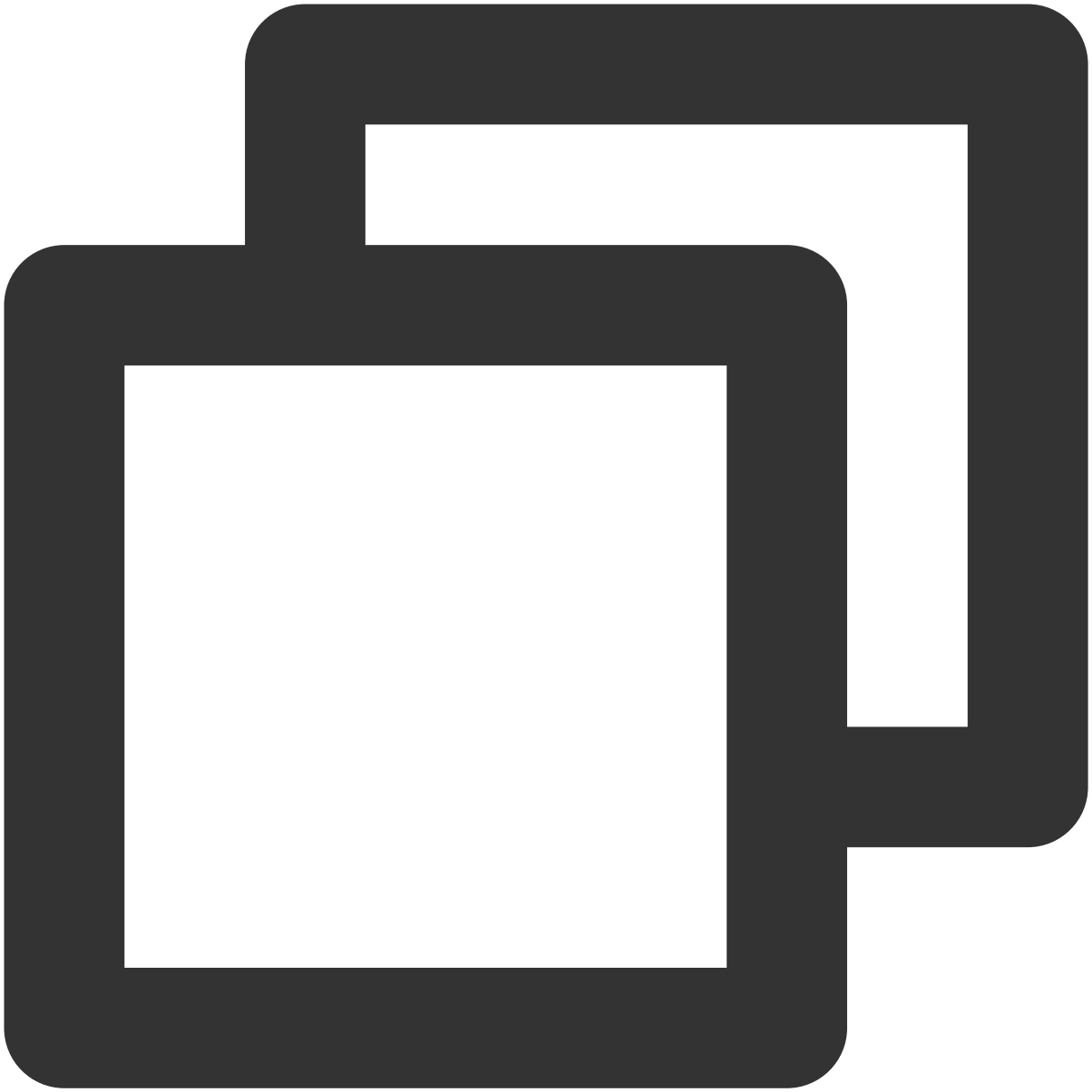
Please make sure you use Shopify CLI 3.0 or higher, if this command does not work.



After your theme app extension is created, you can preview your changes in real time by starting a local development server.

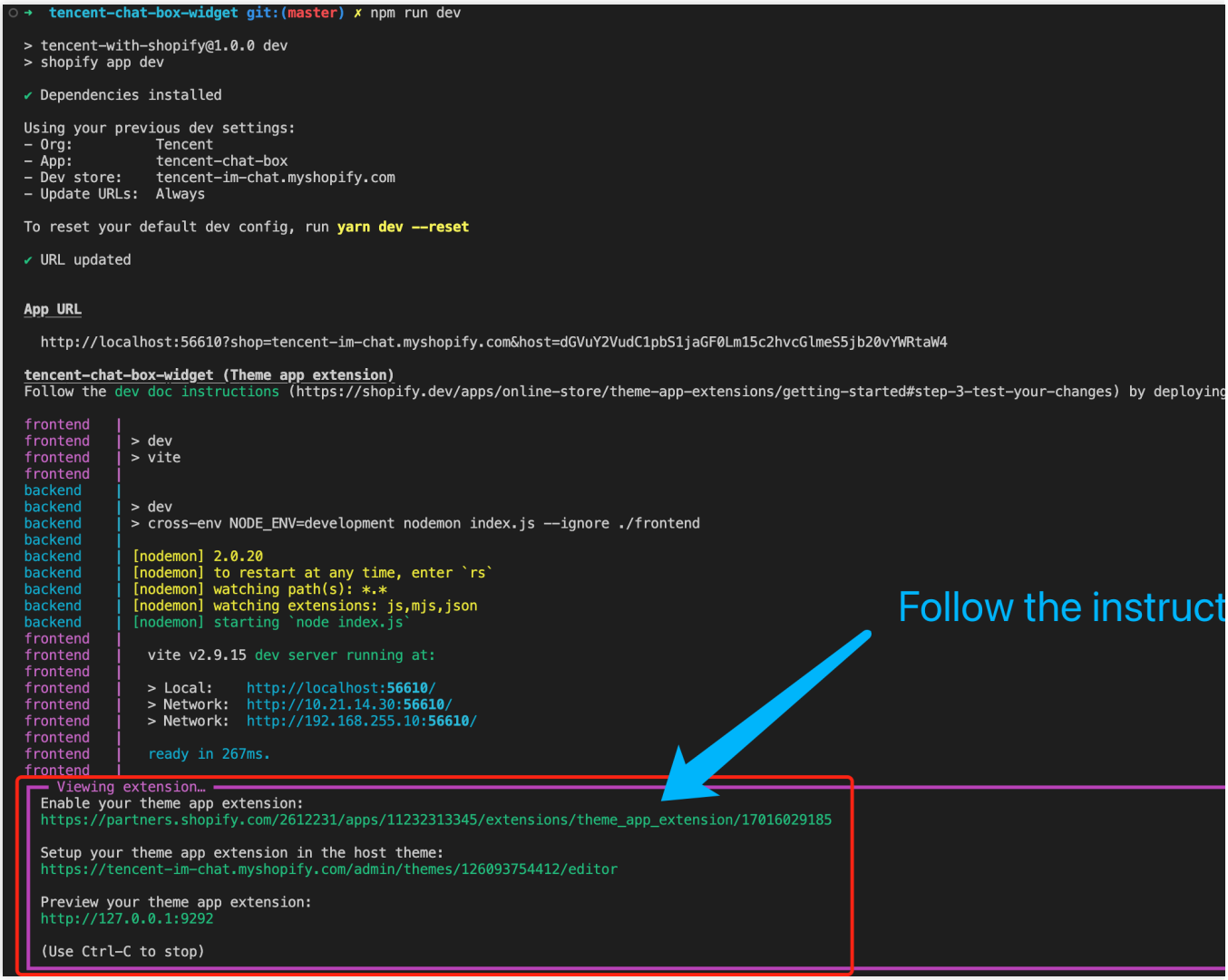
The `dev` command returns a preview URL that reloads local changes, allowing you to preview changes in real time using the store's data. This preview is only available in Google Chrome.

1. Navigate to your app directory.
2. Run one of the following commands:



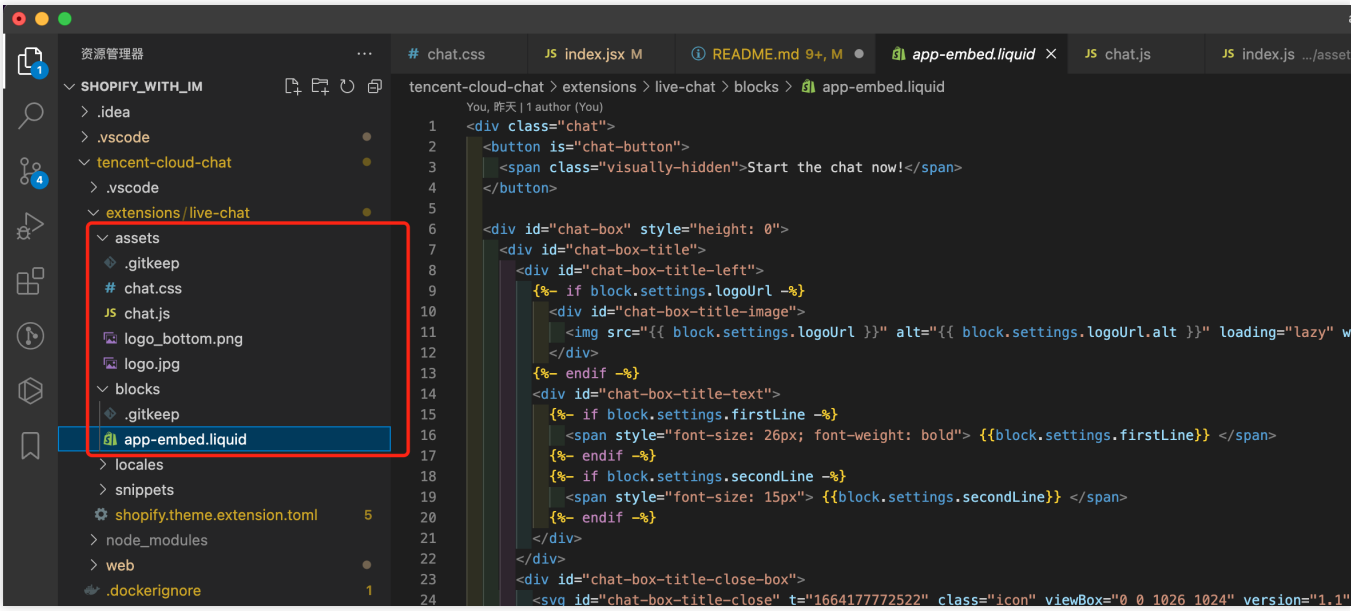
```
npm run dev
```

3. Follow the instructions in the CLI output to enable your theme app extension and set it up in the host theme.
4. Click the URL that's printed at the bottom of the CLI output to preview your extension.



Step 6: Modify the code of extension to build a chat box widget

1. Download our sample code package.
2. Unzip it.
3. Move and replace the `assets` and `blocks` directory to `extensions/(your extension directory)` , and it will be like:



Step7: Start up a server to generate UserSig for each UserID

As you can see in our DEMO (password is `tencentim`), visitors are supposed to start a chat with their email address, and this field will be the user ID for you app of Tencent Cloud Chat.

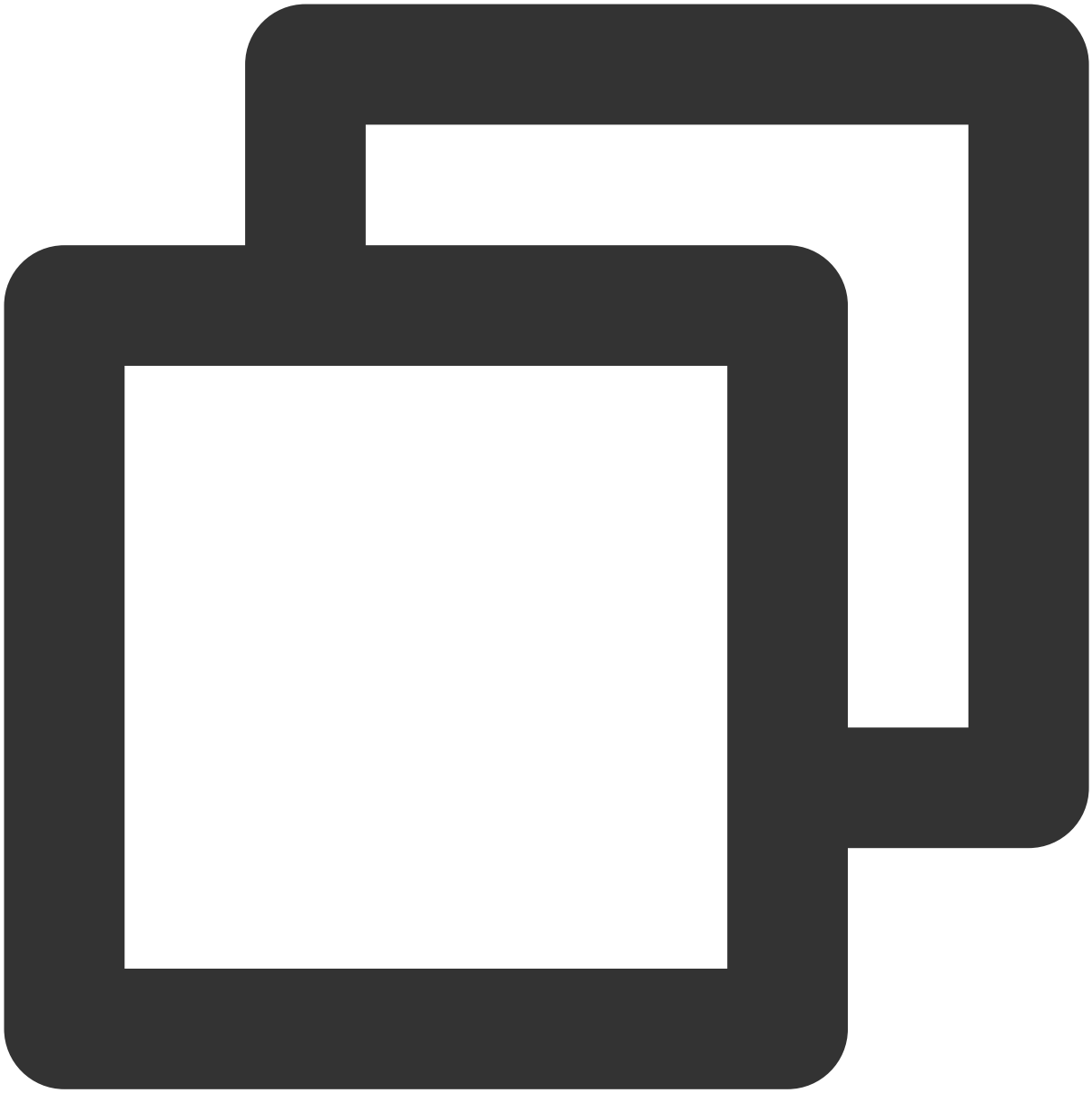
Both `UserID` and `UserSig` are needed for login before chatting, while the `UserSig` is generated with `UserID`, `SECRETKEY` and `SDKAppID` dynamically.

As a result, you should start up a server to generate `UserSig` from `UserID`, while the `SECRETKEY` and `SDKAppID` have been stored on your server for safety purposes.

You could build this service by yourself or using the serverless template we provided.

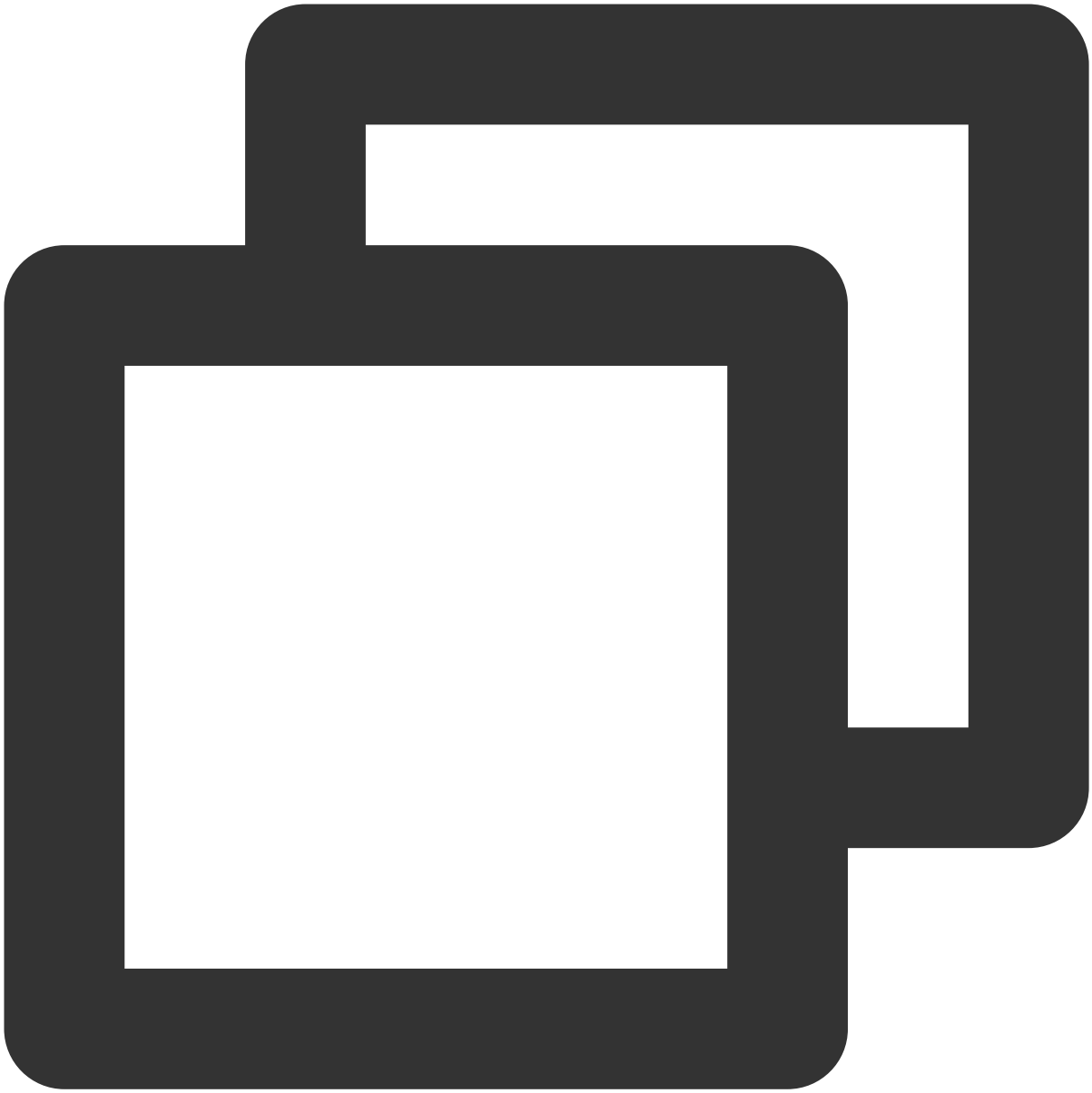
The communication between the chat box and your server should use RESTful API, with POST request.

The following shows request parameters from the chat widget. The `Content-Type` is `application/json`.



```
{
   "userID": "The User ID"
}
```

The response body from your server should be like following:

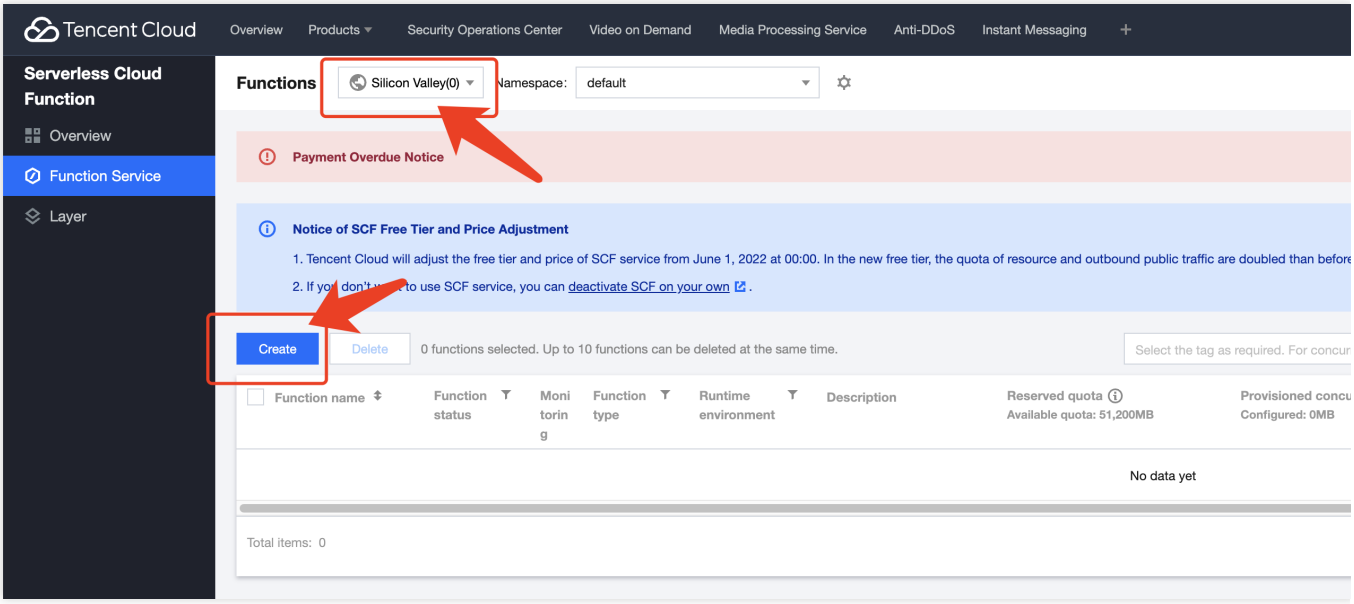


```
{
   "userID": "",
   "userSig": ""
}
```

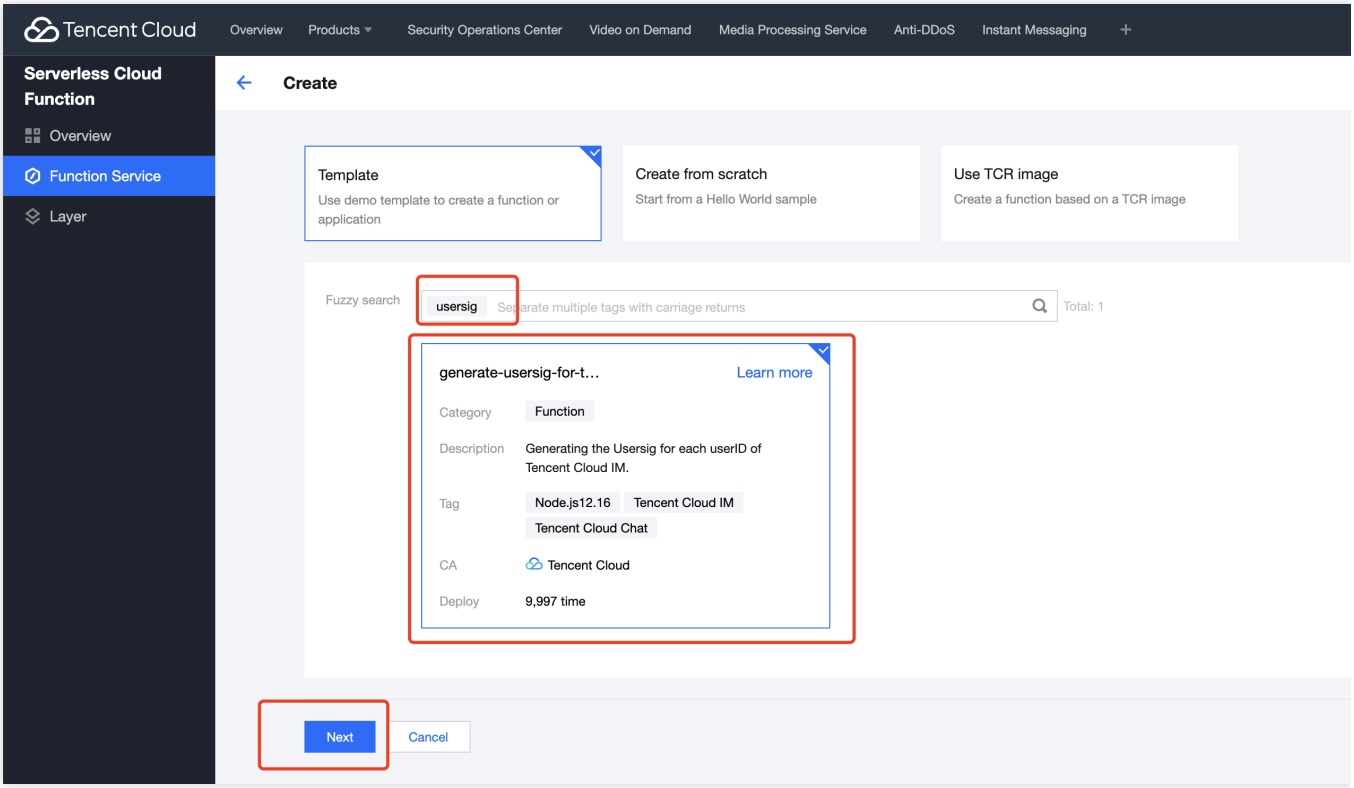
Set up the `UserSig` generation service with serverless function

You are also encouraged to use our template to create a serverless function on Tencent Cloud, if you find complicating to build such a `UserSig` generation service.

1. Navigate to the console of Tencent Cloud Serverless Cloud Function, choose a region and click `Create` .



2. Search by keyword, `usersig` , choose the following one, then click `Next` .

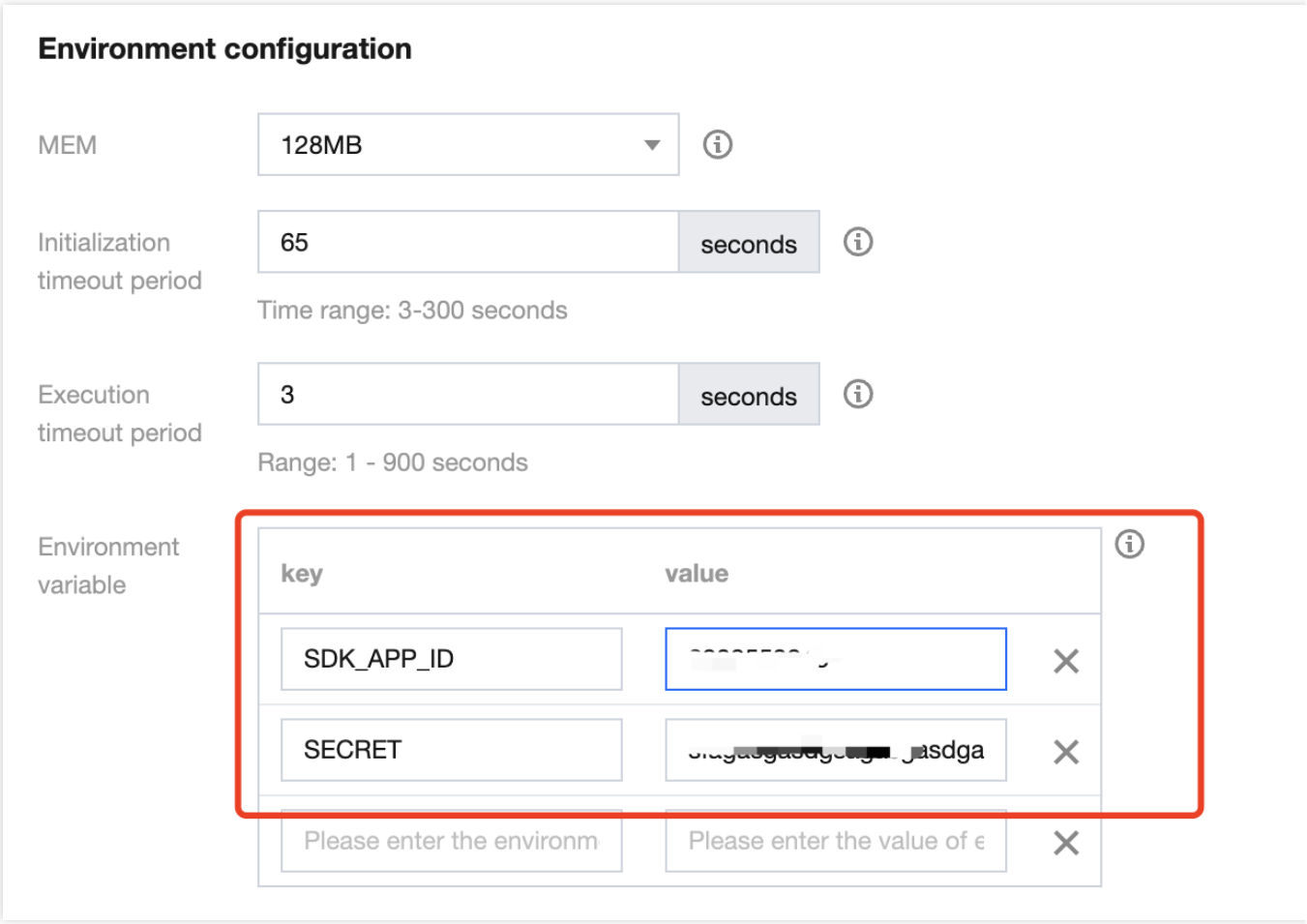


3. Specify the `Function name` based on your needs.

4. Click the `Advanced configuration` module and add the following `Environment configuration` .

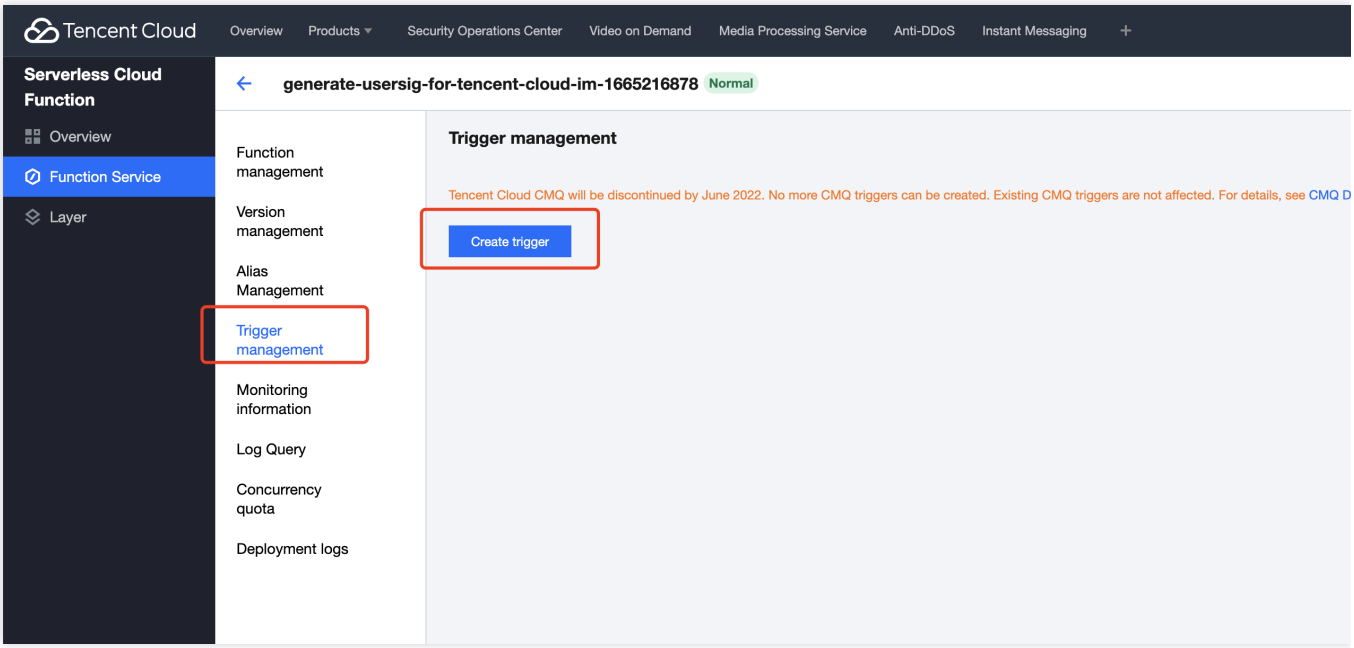
key	value

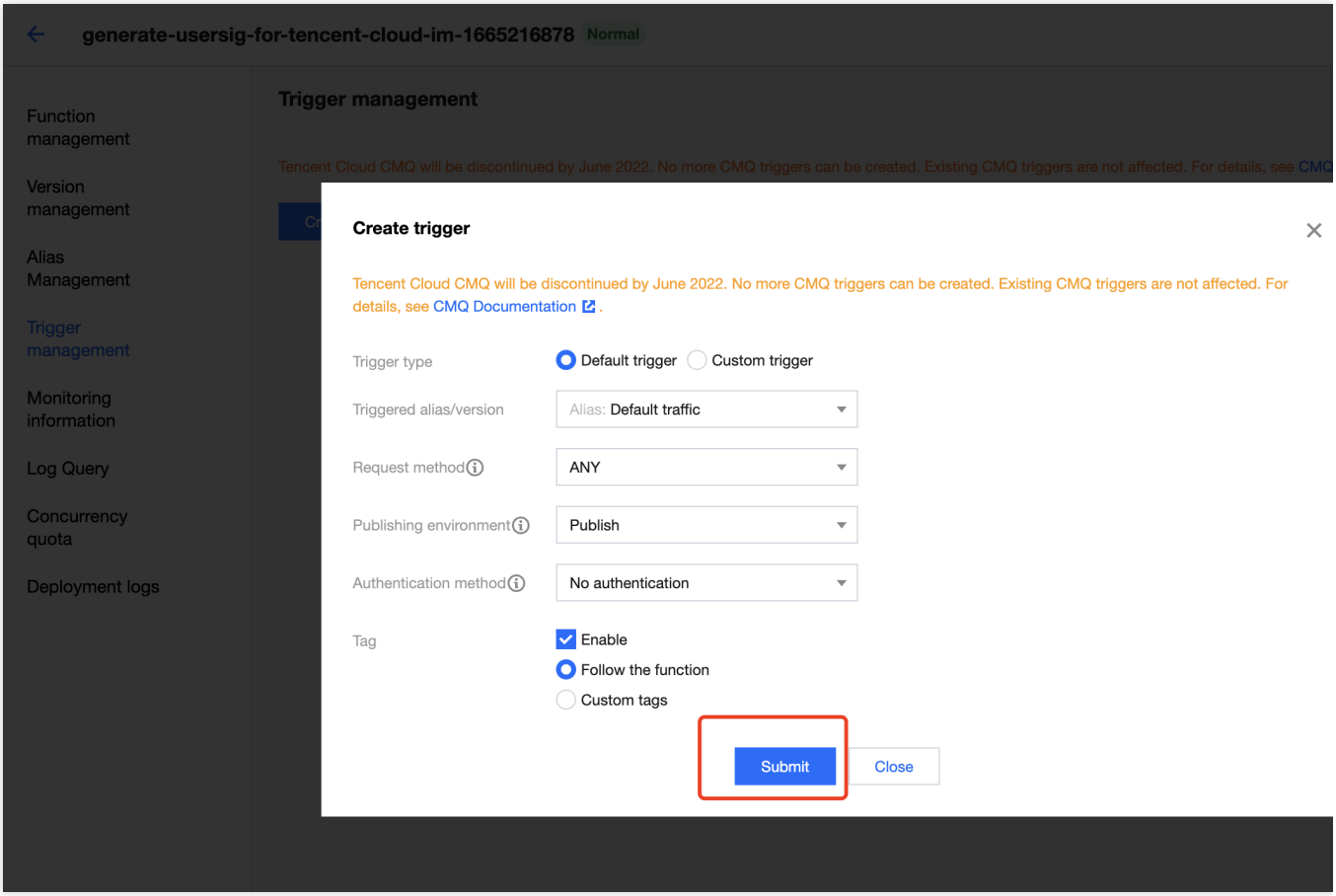
SDK_APP_ID	(Your SDKAppID)
SECRET	(The `Key` shows on the main page of the IM console)



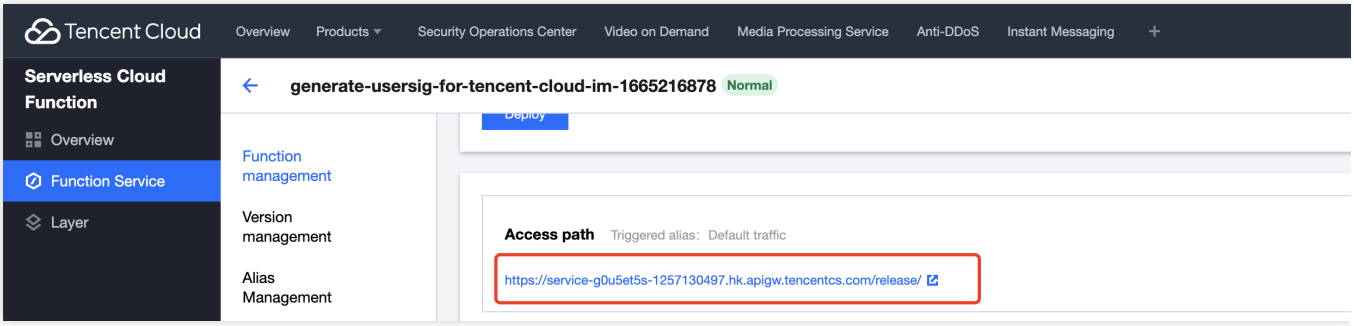
5. After deployment, create a trigger for it by clicking `Trigger management` on the left bar, then click `Create trigger`. Use the default settings on the modal opened, then click `Submit`.

> It might require granting permissions to `API Gateway` first, please follow the instructions to grant.

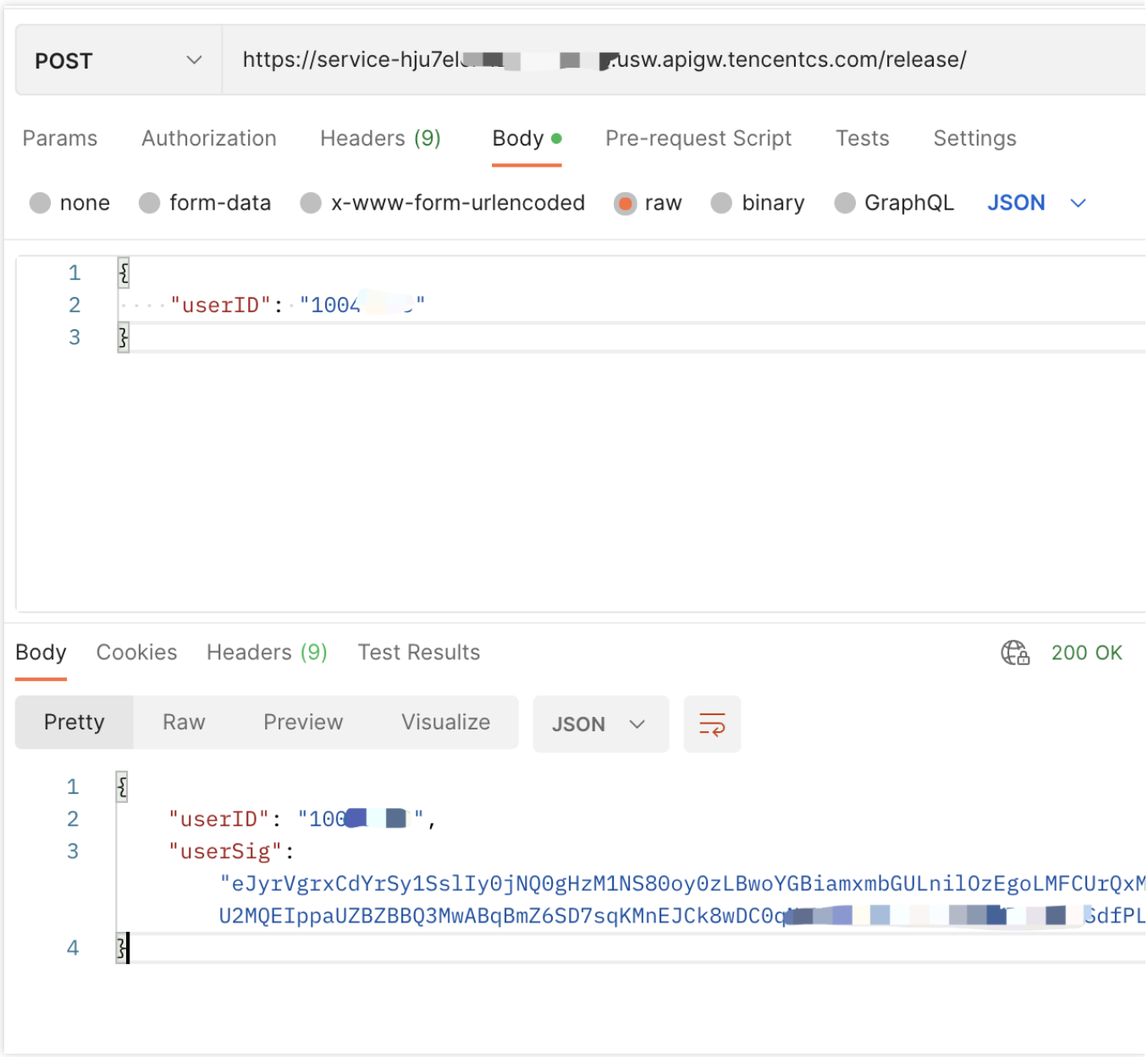




6. Find the URL of the API created from `Function management` => `Function codes` => `Access path`. This URL will be used in the following steps.

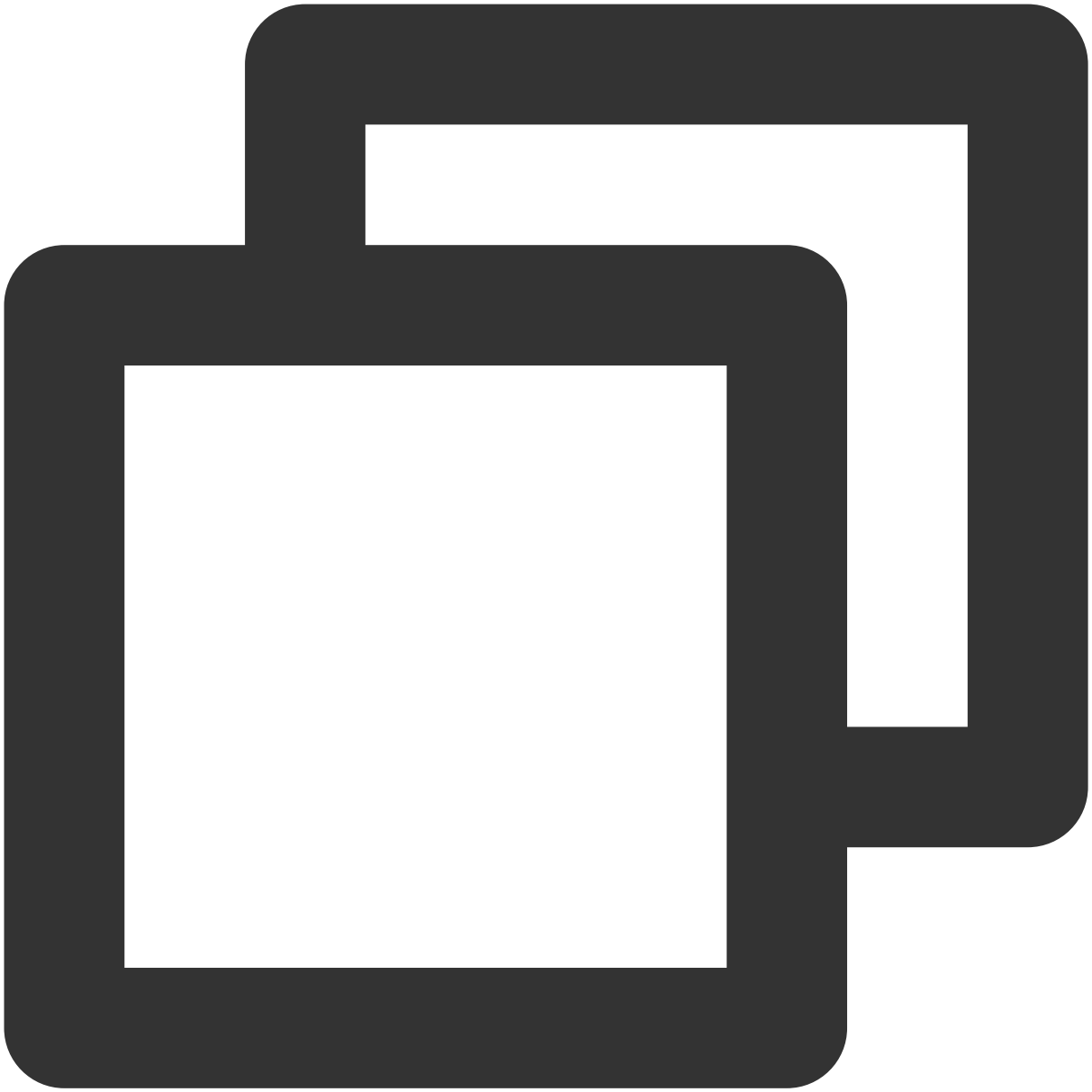


7. [Optional] You can test it by `Postman` .



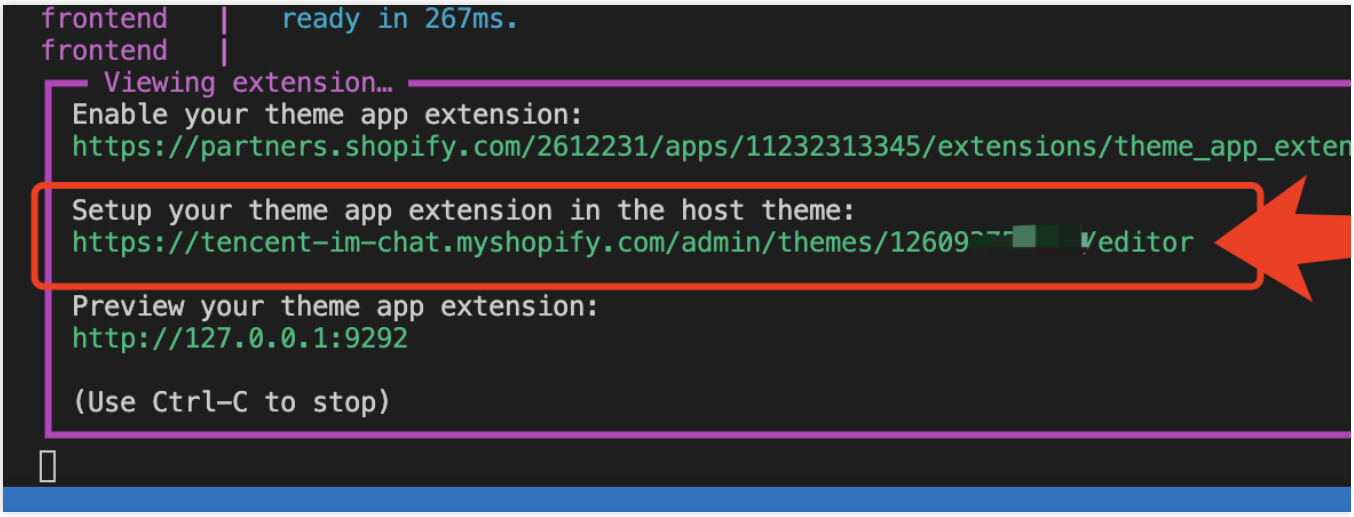
Step 8: Configure this extension

Run the following commands to start the extension:



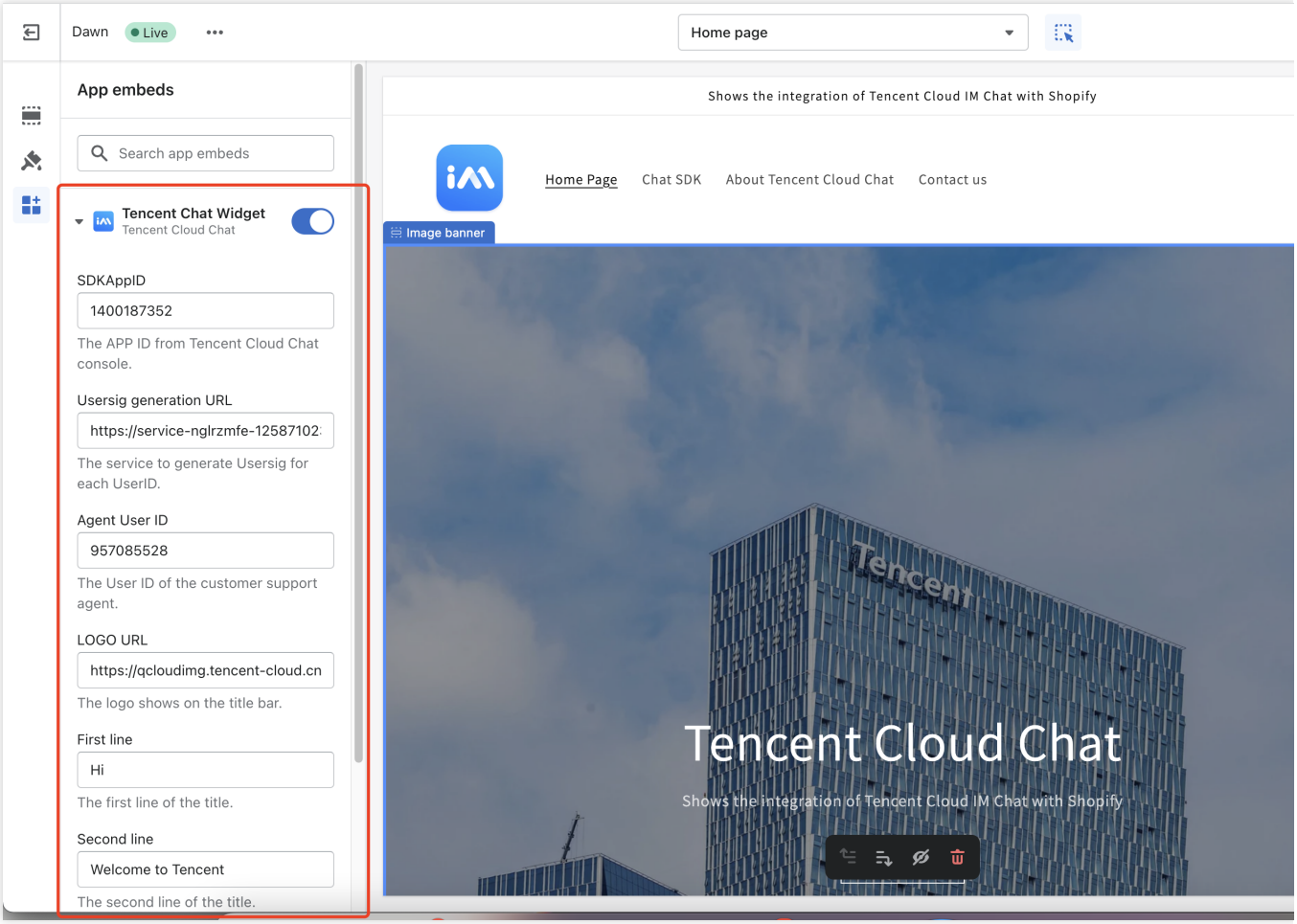
```
npm run dev
```

Click the second URL that's printed at the bottom of the CLI output to set up the extension.



Switch to 'App embeds' on the left menu, and enable 'Tencent Chat Widget'.

Fill in the settings related to it. The detailed introduction for each field shows below the image.

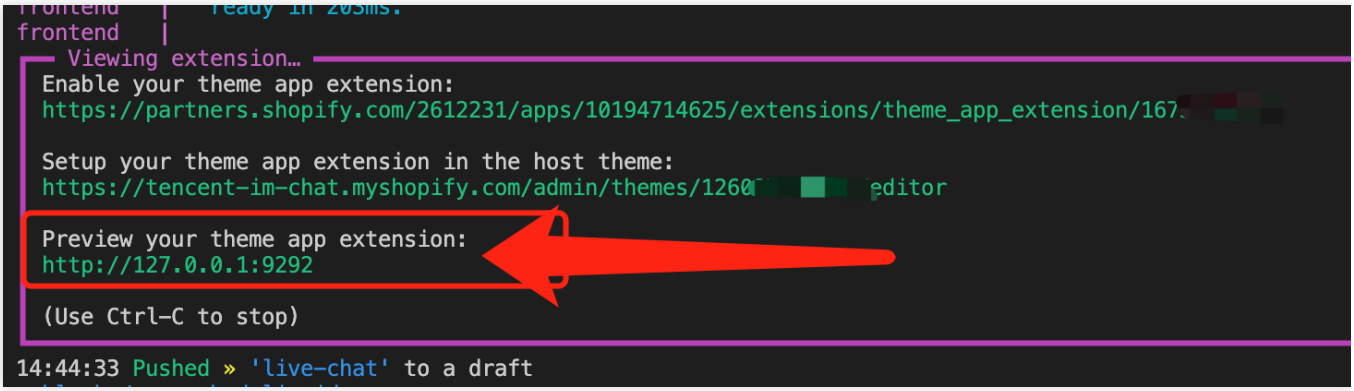


Field	Description	Required

SDKAppID	The `SDKAppID` from Tencent Cloud Chat console.	true
Usersig generation URL	The service to generate Usersig for each UserID. The URL of the API was created from step 7.	true
Agent User ID	The User ID of the customer support agent. The user who visitors chat with.	true
LOGO URL	The logo shows on the title bar.	false
First line	The first line of the title.	false
Second line	The second line of the title.	false

After the configuration, click the `Save` button on the top right.

Now, you can click the third URL that's printed at the bottom of the CLI output to preview the extension.

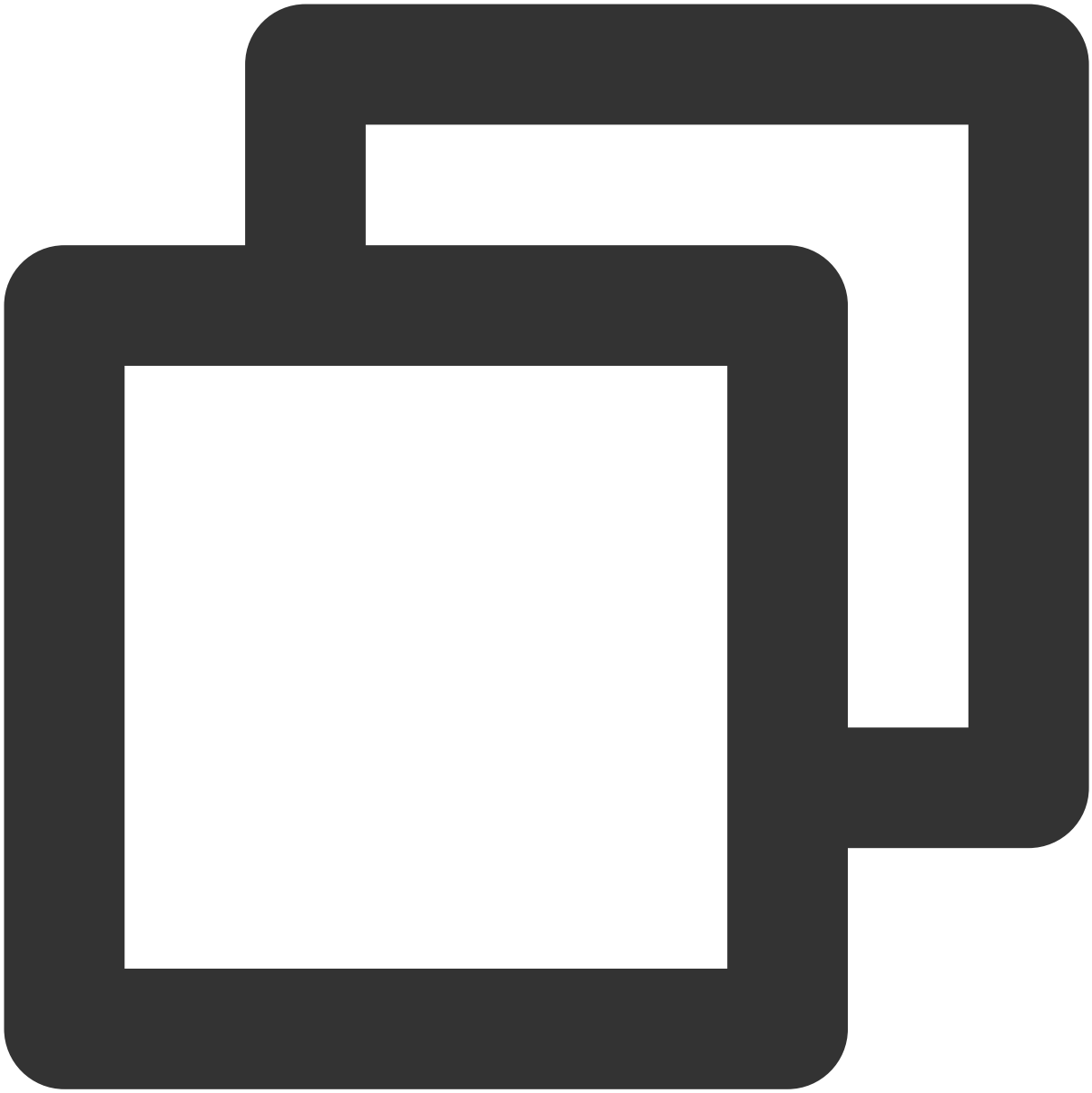


Make sure the chat widget works well. You can refer to our sample codes, if errors are encountered.

Step 9: Deploy the chat widget extension

After your chat widget extension is ready for online stores, you can deploy the latest code and make the app extension public to the development store.

1. Navigate to your app directory.
2. Run the following commands:



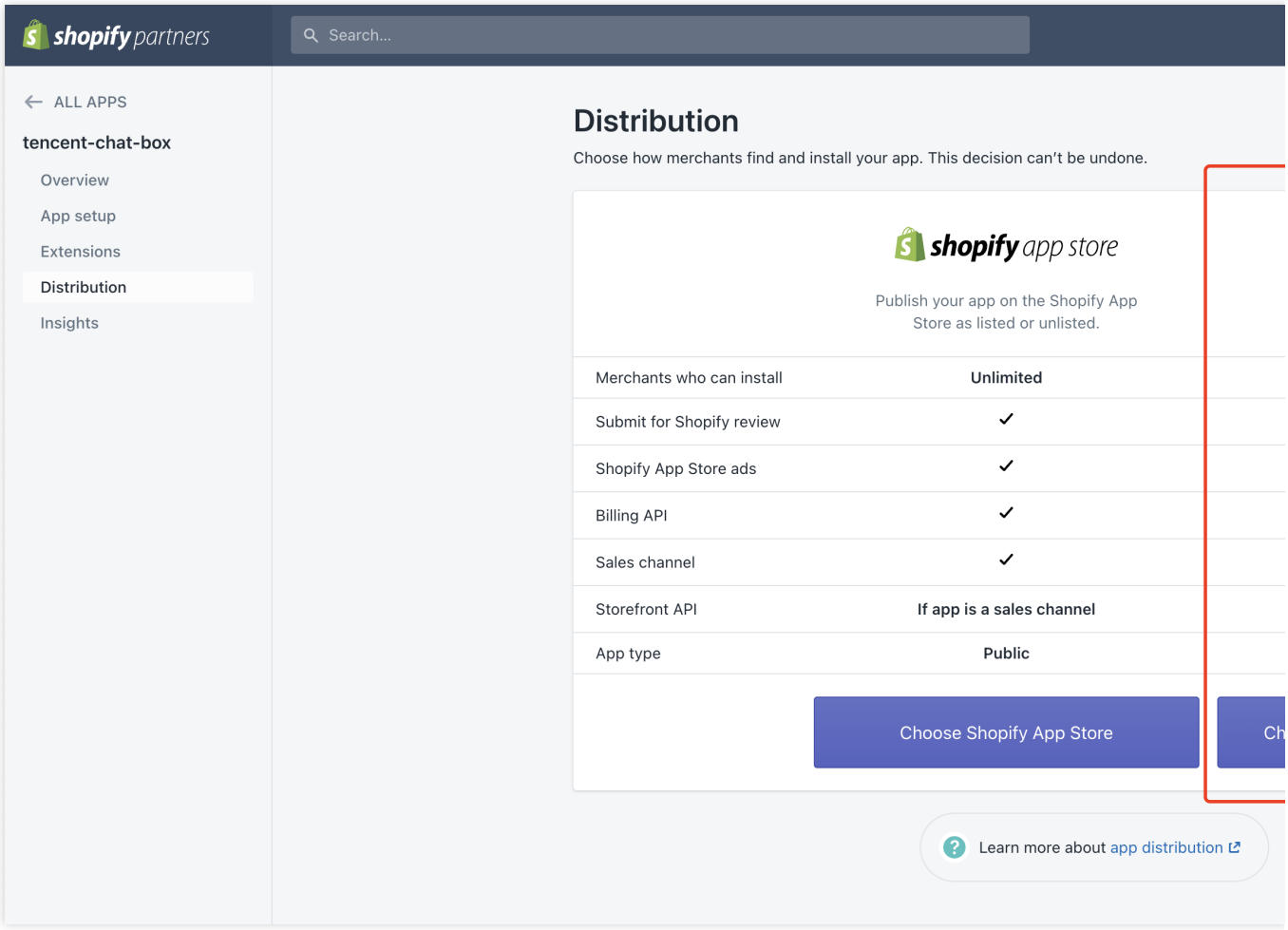
```
npm run deploy
```

3. After you deploy the extension to Shopify, navigate to your app in the Partner Dashboard, click `Extensions` , and click the draft version that you previously created.
4. Click `Create version` to create a version of your theme app extension.
5. Click `Publish` beside the version that you want to publish, and click `Publish` in the popup modal to confirm.

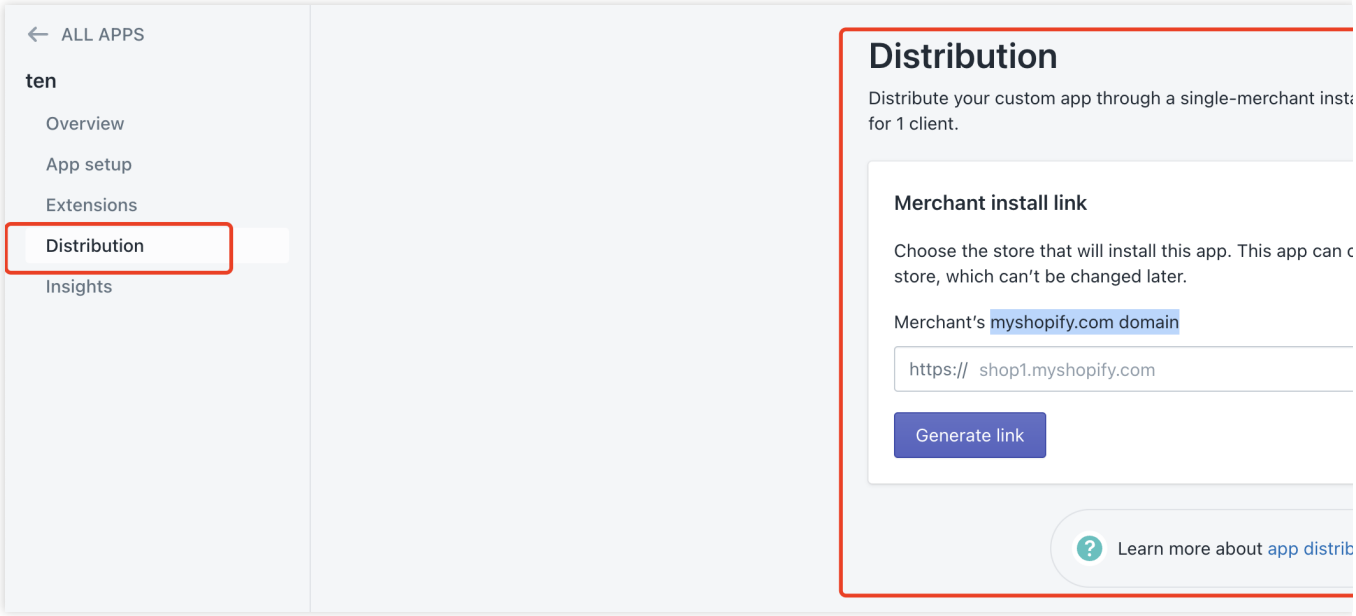
Step 10: Install this app to your online store

1. From the Shopify Partner Dashboard, go to Apps and then select your newly created app from the list.

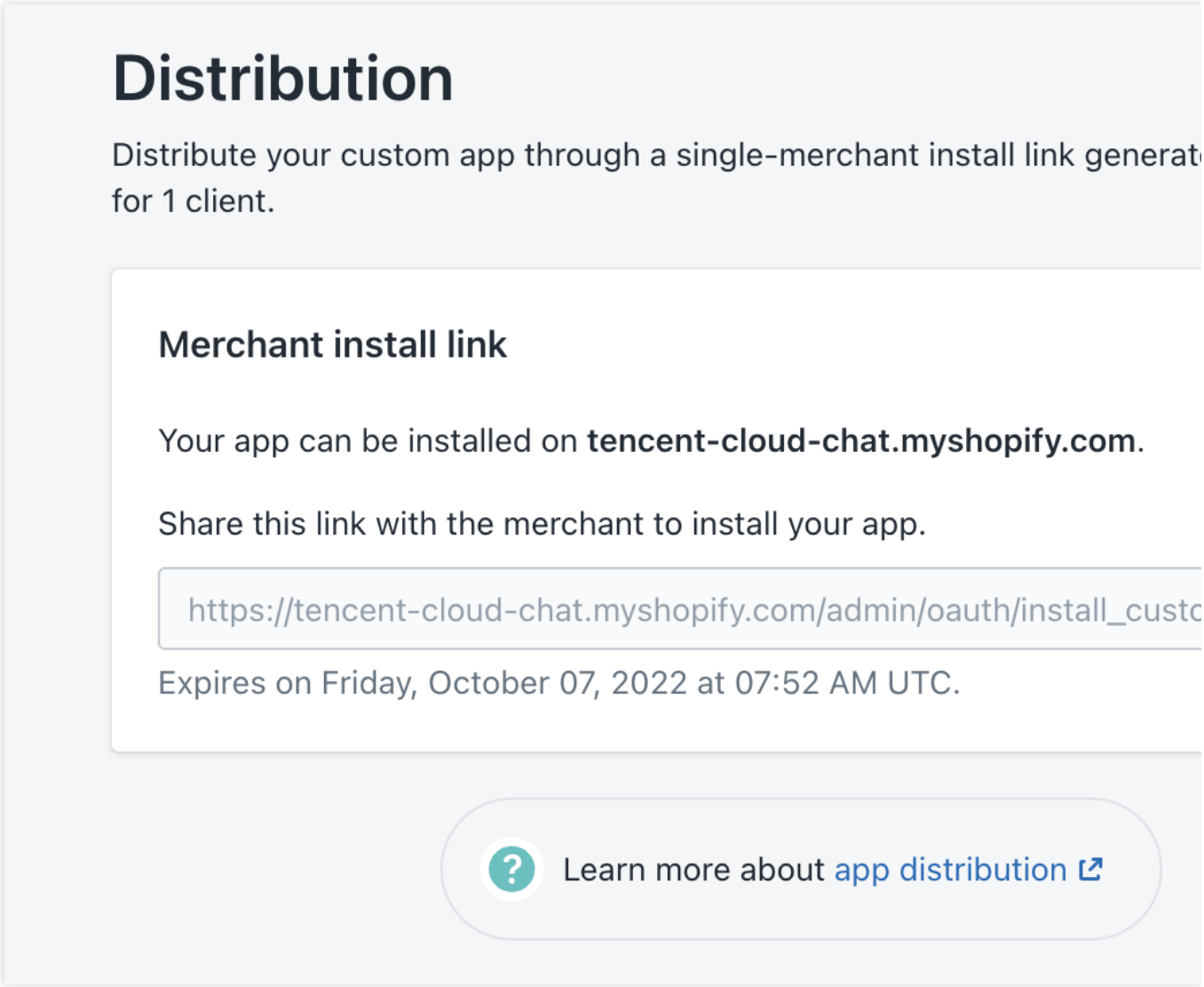
2. In the sidebar, click **Distribution**.
3. Select "Single-merchant install link" as the distribution method.



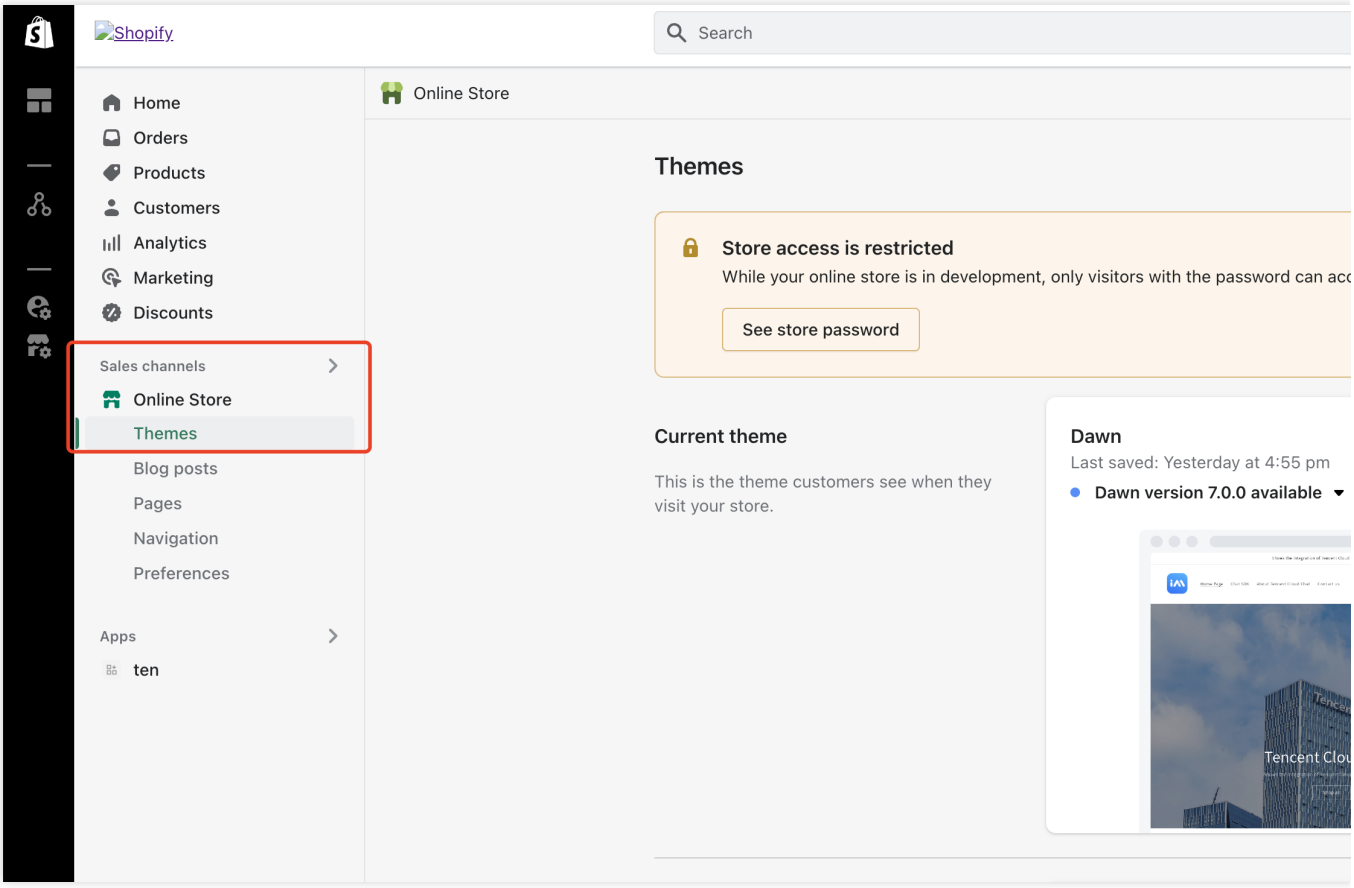
4. Generate the install link for your Shopify store, by entering your `myshopify.com` domain here.



5. Use the generated link to install the app in your current store.



6. Enable and configure the widget on the `Themes` section of your admin dashboard. The steps of enabling and configuring the widget can refer to step 8.



Conclusion

That's all you need to add a Tencent Cloud Chat widget to your Shopify store.

If there's anything unclear or you have more thinkings about the integration with Shopify, feel free to contact us!

Discord 구현 가이드

최종 업데이트 날짜: : 2023-03-28 10:09:31

Discord 개요

Discord는 커뮤니티를 위해 설계된 무료 온라인 실시간 통화 소프트웨어 및 디지털 배포 플랫폼입니다. 게임, 교육 및 비즈니스 분야의 사용자는 소프트웨어의 채팅 채널에서 메시지, 이미지, 영상 및 음성을 통해 서로 통신할 수 있습니다.

Discord 개념

서버

Discord에는 서버(커뮤니티와 유사)라는 일반적인 통신 소프트웨어 그룹과 다른 일종의 그룹 채팅이 있으며, 서버 소유자는 서버에서 자신의 커뮤니티를 만들 수 있습니다.

채널

서버에서 음성 및 텍스트 채널을 포함한 채팅 채널을 만들 수 있습니다. 음성 채널은 게임 방송, 채팅 등을 위해 사용할 수 있습니다. 다양한 권한을 ID 그룹과 통합하도록 채널을 설정하여 Discord 커뮤니티 시스템을 더욱 다양하게 만들 수 있습니다.

스레드

사용자는 스레드에서 특정 토픽에 대해 토론할 수 있습니다.

준비 작업

Tencent Cloud IM 애플리케이션 생성

이 튜토리얼은 Tencent Cloud IM을 기반으로 합니다. 따라서 Tencent Cloud IM 콘솔에서 애플리케이션을 생성해야 합니다. 아래 이미지를 참고하십시오.

애플리케이션이 성공적으로 생성되면 IM 콘솔 애플리케이션의 기본 정보 페이지에서 애플리케이션의 기본 정보를 확인할 수 있습니다.

관련 구성 및 기능 이해

Tencent Cloud IM을 사용하여 Discord 기능을 구현하려면 Tencent Cloud IM과 관련된 기본 개념과 이 튜토리얼의 뒷부분에서 언급되는 몇 가지 고유 명사를 미리 이해해야 합니다. 다음을 포함하되 이에 국한되지 않습니다:

- SDKAppID: Tencent Cloud IM은 각 애플리케이션에 SDKAppID를 할당합니다. 애플리케이션의 SDKAppID는 애플리케이션이 콘솔에서 생성된 후 애플리케이션 세부 정보 페이지에서 볼 수 있으며 IM SDK를 초기화하고 사용자 로그인 티켓을 계산할 때 사용됩니다. 자세한 내용은 UISDK 초기화 및 로그인을 참고하십시오.
- 키: 애플리케이션의 키는 Tencent Cloud IM 콘솔 애플리케이션 세부 정보 페이지에서 볼 수 있으며 사용자의 SDK 로그인 티켓을 계산하는 데 사용됩니다.
- 사용자 계정: Tencent Cloud IM 계정이 있는 사용자만 Tencent Cloud IM에 로그인할 수 있습니다. 사용자가 클라이언트 SDK를 통해 성공적으로 Login하면 IM 백엔드가 사용자의 IM 계정을 자동으로 생성합니다. 또한 IM에서 제공하는 서버 API를 사용하여 Importing a Single Account할 수 있습니다.
- 그룹: 지금까지 IM은 사용자가 메시지를 보내고 받을 수 있는 다양한 시나리오에 대해 5가지 유형의 그룹을 제공합니다.
- 콜백 구성: IM에서 제공하는 클라이언트 및 서버 API를 사전에 통합할 수 있으며 IM은 특정 비즈니스 로직이 트리거될 때 필요한 정보를 서버에 자동으로 반환합니다. 채팅 콘솔의 콜백 구성 모듈에서 관련 구성을 완료하기만 하면 됩니다. IM은 다양한 콜백 구성을 제공하고 콜백 이벤트에 대한 높은 안정성을 보장합니다. 콜백를 사용하여 다양한 사용자 지정 요구 사항을 구현할 수 있습니다.
- 사용자 지정 필드: 기본적으로 Tencent Cloud IM은 개발자의 요구 사항을 대부분 충족하는 필드를 제공합니다. 또한 확장 필드에 대한 사용자의 요구를 충족하기 위해 각 모듈에 대해 다음과 같은 사용자 정의 필드를 제공합니다.
  - 사용자 지정 사용자 필드
  - 사용자 지정 친구 필드
  - 사용자 지정 그룹 필드
  - 사용자 지정 구성원 필드

> 설명 :
> 사용자 지정 필드를 사용하려면 콘솔에서 구성한 다음 SDK\API를 통해 읽고 쓸 수 있습니다.

클라이언트&서버 SDK 통합

Discord 관련 기능을 구현하려면 Tencent Cloud IMSDK를 통합해야 합니다. IM은 다양하고 사용하기 쉬운 SDK와 서버 API를 제공합니다. 동일한 SDKAppID로 애플리케이션에 로그인하면 메시지가 클라이언트 간에 동기화됩니다. 비즈니스 요구 사항 시나리오 및 기술 스택에 따라 적절한 SDK를 선택합니다.

Discord 기능 분석

상기 이미지에서 볼 수 있듯이 Discord 기능에는 서버, 채널 및 스레드가 포함됩니다. 각기 다른 콘텐츠에는 다른 서버가 사용됩니다. 예를 들어 Honor of Kings 서버와 Game for Peace 서버가 있습니다. 문자 채널, 음성 채널, 알림 채널 등 다양한 형태의 채널을 서버에서 생성할 수 있으며 사용자는 채널에서 서로 통신합니다. 특정 토픽에 대해 더 많은

아이디어가 있는 경우 추가 토론을 위한 스레드를 만들 수 있습니다. 다음은 Tencent Cloud IM을 통해 이러한 주요 Discord 기능을 구현하는 방법을 하나씩 소개합니다.

> 설명 :
> **코드 데모의 경우 Android 클라이언트(Java SDK)를 예로 사용합니다. 다른 SDK 버전의 API 호출은 통합 솔루션(UI 없음)을 참고하십시오.**

서버

서버 생성

분석 결과 Discord 서버 목록의 다음과 같은 특징을 알 수 있습니다.

1. 서버에는 방대한 수의 사용자가 있을 수 있습니다.
2. 사용자는 실제로 서버에서 서로 통신하지 않습니다. 대신 서버의 채널이나 스레드에서만 채팅합니다.
3. 서버의 채팅 메시지 기록은 로밍 서버에 저장해야 합니다.
4. 자유롭게 액세스할 수 있습니다.
5. 서버에서 채널을 생성할 수 있습니다.

IM은 5가지 유형의 그룹을 제공하지만 커뮤니티 그룹(Community)만이 Discord 서버의 특성을 충족합니다. IM 커뮤니티 그룹은 사용자가 자유롭게 가입 및 탈퇴할 수 있으며 최대 10만 명의 구성원을 지원하고 메시지 기록을 저장합니다. 그룹에 가입하려면 사용자는 그룹 ID를 검색하고 신청해야 하며, 신청서는 관리자의 승인 없이 그룹에 가입할 수 있습니다.

createGroup API를 사용하여 서버(그룹)를 생성할 수 있습니다. 그룹 유형은 Community 이고 setSupportTopic이 true 로 설정되어야 서버에서 채널이 생성될 수 있습니다. 다음은 샘플 코드입니다.

```
V2TIMGroupInfo groupinfo = new V2TIMGroupInfo();
groupinfo.setGroupName("테스트 서버");
groupinfo.setSupportTopic(true);
// 초기 그룹 구성원
List<V2TIMCreateGroupMemberInfo> memberList = new LinkedList<V2TIMCreateGroupMemb
erInfo>();
// 서버 프로필 사진과 같은 기타 구성
V2TIMManager.getGroupManager().createGroup(groupinfo, memberList, new V2TIMValueC
allback<String>() {
@Override
public void onError(int i, String s) {
// 생성 실패
}

@Override
```

```
public void onSuccess(String s) {
// 생성 성공, 서버 ID가 반환됩니다
}
});
```

API가 성공적으로 호출되면 onSuccess 콜백에 서버 ID가 반환되며 이는 나중에 서버에서 채널을 생성할 때 사용됩니다.

주의 :
IM에서 제공하는 서버 생성 API를 이용해도 됩니다. 주요 매개변수는 다음과 같습니다.

```
{
"Type": "Community", // 그룹 유형(필수)
"Name": "TestCommunityGroup", // 그룹 이름(필수)
"SupportTopic": 1 // 토픽 옵션 지원 여부. 유효한 값: 1 yes, 0 no
}
```

서버 목록

Discord의 맨 왼쪽에는 현재 사용자가 가입한 서버 목록이 있습니다. 커뮤니티 시나리오의 경우, Tencent Cloud IM은 서버 목록 조회 전용 API를 제공합니다.

```
V2TIMManager.getGroupManager().getJoinedCommunityList(new V2TIMValueCallback<List
<V2TIMGroupInfo>>() {
@Override
public void onSuccess(List<V2TIMGroupInfo> v2TIMGroupInfos) {
// 서버 목록 가져오기 성공, 반환된 List<V2TIMGroupInfo>에 서버 목록의 기본 정보가 제공됩니
다.
}

@Override
public void onError(int i, String s) {
// 서비스 목록 가져오기 실패
}
});
```

반환된 V2TIMGroupInfo는 기본 서버 정보를 제공합니다. 그러나 반환된 서버 정보에는 읽지 않은 메시지 수 및 사용자 지정 상태와 같은 정보가 포함되지 않습니다. Discord에서와 동일한 효과를 얻으려면 IM에서 제공하는 다른 API를 사용하여 서버 목록에 대한 읽지 않은 메시지 수를 구현해야 합니다. 본 문서의 뒷부분에서 이에 대해 설명합니다.

서버 카테고리

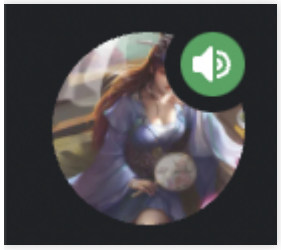
서버가 생성되면 기본 카테고리가 생성됩니다. 서버가 생성된 후 새 카테고리를 생성할 수 있습니다. Tencent Cloud IM에서 서버는 본질적으로 커뮤니티입니다. 따라서 커뮤니티 그룹에 대한 사용자 정의 필드를 설정하여 서버 유형 기능을 구현할 수 있습니다. 사용자 정의 그룹 필드를 사용하려면 아래 2단계를 수행하십시오.

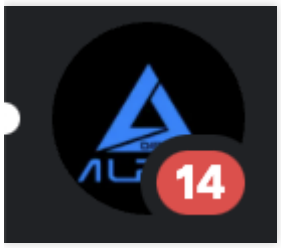
1. 콘솔에서 사용자 정의 필드 key를 활성화합니다.
2. 클라이언트 SDK\서버 API를 사용하여 사용자 지정 필드를 읽고 씁니다.

서버 API와 클라이언트 SDK를 이용하여 사용자 지정 그룹 필드를 설정합니다.

> 주의 :
> 커뮤니티 그룹 기능은 프리미엄 에디션에서만 사용할 수 있습니다. 커뮤니티 그룹에 대한 사용자 지정 필드를 설정하기 전에 프리미엄 에디션을 구입해야 합니다.

서버의 읽지 않은 메시지 수&사용자 지정 상태 표시





이전에 언급했듯이, 가입한 서버 목록을 가져오는 API는 읽지 않은 메시지 수 및 서버 상태와 같은 정보를 반환하지 않습니다. 이 데이터를 가져오는 것뿐만 아니라 이 데이터의 변경을 수신하여 클라이언트 UI를 적시에 업데이트해야 합니다. 서버는 IM 커뮤니티 그룹에 의해 구현되며, IM에서 커뮤니티 그룹은 대화를 생성하지 않으므로 모든 공개 및 비공개 채널 대화의 읽지 않은 메시지 수의 합계를 계산해야 합니다. V2TIMTopicInfo의 getUnreadCount API를 사용하여 공개 채널의 읽지 않은 메시지 수를 가져오고, 비공개 채널의 읽지 않은 메시지 수는 work 그룹을 통해 구현되기 때문에, getConversation API를 사용하여 가져옵니다.

```
// 공개 채널
List<String> conversationIDList = new LinkedList();
```

```
conversationIDList.add("GROUP_$GROUPID");
V2TIMManager.getConversationManager().getConversationList(conversationIDList, new
V2TIMValueCallback<List<V2TIMConversation>>() {
@Override
public void onError(int i, String s) {
// 서버의 대화 정보 가져오기 실패
}

@Override
public void onSuccess(V2TIMConversationList List<V2TIMConversation>) {
// 서버의 대화 정보 가져오기 완료
}
});
// 비공개 채널
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList, new V2TIMVa
lueCallback<List<V2TIMTopicInfoResult>>() {
@Override
public void onSuccess(List<V2TIMTopicInfoResult> v2TIMTopicInfoResults) {

}

@Override
public void onError(int i, String s) {
}
});
```

서버에 대한 사용자 지정 상태 기능을 구현하기 위해 setConversationCustomData API를 사용하여 사용자 지정 서버 대화 데이터를 설정할 수 있습니다.

```
List<String> conversationIDList = new LinkedList();
String customData = "통화 중"
V2TIMManager.getConversationManager().setConversationCustomData(conversationIDLis
t, customData, new V2TIMValueCallback<List<V2TIMConversationOperationResult>>() {
@Override
public void onSuccess(List<V2TIMConversationOperationResult> v2TIMConversationOpe
rationResults) {
// 사용자 지정 그룹 대화 데이터 설정 성공
}

@Override
public void onError(int i, String s) {
// 사용자 지정 그룹 대화 데이터 설정 실패
}
});
```

서버 대화와 관련된 데이터가 변경되면 클라이언트는 UI 표시 업데이트를 시도해야 합니다. 서버 대화 변경 사항을 수신하기 위해 IM은 해당 이벤트 리스너 함수 addConversationListener를 제공합니다. 이 콜백 함수는 다음과 같은 경우에 트리거됩니다.

1. 서버 메시지 추가, 삭제 또는 수정
2. 읽지 않은 서버 메시지 수 변경
3. 사용자 지정 서버 정보 수정
4. 서버 상단 고정
5. 서버 메시지 수신 구성 수정
6. 서버 마크 수정
7. 서버 그룹 수정
8. ...

```
V2TIMConversationListener conversationLister = new V2TIMConversationListener() {
@Override
public void onSyncServerStart() {
}

@Override
public void onSyncServerFinish() {
}

@Override
public void onSyncServerFailed() {
}

@Override
public void onNewConversation(List<V2TIMConversation> conversationList) {
}

@Override
public void onConversationChanged(List<V2TIMConversation> conversationList) {
}
@Override
public void onTotalUnreadMessageCountChanged(long totalUnreadCount) {
}

@Override
public void onConversationGroupCreated(String groupName, List<V2TIMConversation>
conversationList) {
}

@Override
public void onConversationGroupDeleted(String groupName) {
```

```
}

@Override
public void onConversationGroupNameChanged(String oldName, String newName) {
}

@Override
public void onConversationsAddedToGroup(String groupName, List<V2TIMConversation>
conversationList) {、
}

@Override
public void onConversationsDeletedFromGroup(String groupName, List<V2TIMConversat
ion> conversationList) {
}
}
V2TIMManager.getConversationManager().addConversationListener(conversationListe
r);
```

상기 내용을 요약하면, getJoinedCommunityList 및 getConversationList API와 addConversationListener 콜백을 사용하여 Discord의 서버 목록을 표시 기능을 구현할 수 있습니다.

채널

서버에 여러 채널을 생성할 수 있습니다. 아래 이미지와 같이 서버 아래에 4개의 채널이 생성되고 4개의 채널이 2개의 카테고리에 배치됩니다.

사용자는 다른 사람을 채널에 초대하고 채널의 기본 설정을 수정할 수 있습니다. 사용자는 대부분의 채팅을 채널 내에서 진행하므로, 채널 기능은 Discord에서 가장 중요합니다. Discord의 채널 기능은 Tencent Cloud IM의 토픽 기능에 해당합니다. IM의 커뮤니티 그룹은 그룹 내 토픽 생성 기능을 제공합니다.

기본 채널

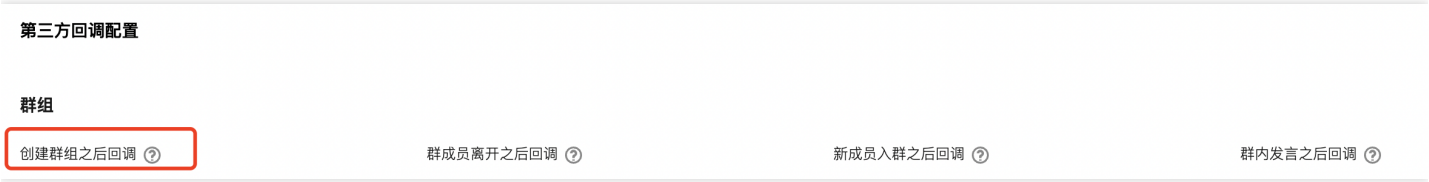
서버가 생성되면 Discord는 서버에 대해 4개의 기본 채널을 생성합니다. Tencent Cloud IM도 이러한 기능을 구현할 수 있습니다. 프로세스는 다음과 같습니다.

1. 그룹 생성 후 콜백을 통해 서버 생성 성공을 비즈니스 서버에 알립니다.
2. 비즈니스측은 생성된 그룹이 커뮤니티 그룹인지 판단하여 다른 그룹과 관련된 비즈니스에 영향을 미치지 않도록 합니다.
3. 서버 속성을 기반으로 서버에서 Creating Topic합니다.

서버 토픽 생성 매개변수는 다음과 같습니다.

```
{
"GroupId": "@TGS#_@TGS#cQVLVHIM62CJ", // 토픽의 그룹 ID(필수)
"TopicId": "@TGS#_@TGS#cQVLVHIM62CJ@TOPIC#_TestTopic", // 사용자 지정 토픽 ID(옵션)
```

```
"TopicName": "TestTopic", // 토픽 이름(필수)
"From_Account": "1400187352", // 토픽을 생성하는 구성원
"CustomString": "This is a custom string", // 사용자 지정 문자열
"FaceUrl": "http://this.is.face.url", // 토픽 프로필 사진 URL(옵션)
"Notification": "This is topic Notification", // 토픽 알림(옵션)
"Introduction": "This is topic Introduction" // 토픽 소개(옵션)
}
```



채널 생성

Discord 클라이언트에서 사용자는 아래 이미지와 같이 다양한 채널 카테고리 아래에 채널을 만들 수 있습니다.

Discord에서의 채널 생성은 IM의 커뮤니티 그룹에서의 토픽 생성과 같습니다. IM에서 토픽 생성 시 토픽의 카테고리와 기본 정보를 설정할 수 있습니다.

createTopicInCommunity API를 사용하여 관련 기능을 구현할 수 있습니다.

```
String groupID = "서버 ID"
V2TIMTopicInfo info = new V2TIMTopicInfo();
info.setCustomString("{'categray':'게임','type':'text'}") // // 채널 카테고리 및 유형 설정
// V2TIMTopicInfo 기본 정보 설정
V2TIMManager.getGroupManager().createTopicInCommunity(groupID, info, new V2TIMValueCallback<String>() {
@Override
public void onSuccess(String s) {
// 채널 생성 성공
}

@Override
public void onError(int i, String s) {
// 채널 생성 실패
}
});
```

채널 생성 시 V2TIMTopicInfo 메소드를 호출하여 채널 정보를 설정하고 setCustomString API를 호출하여 채널 카테고리 및 유형을 설정할 수 있습니다.

채널을 생성할 때 비공개 채널 여부를 지정할 수 있습니다. 비공개 채널은 일반 채널과 다릅니다.

1. 사용자가 서버에 가입한 후, 비공개 채널에 가입되지 않습니다.
2. 사용자는 서버 관리자가 초대한 경우에만 비공개 채널에 참여할 수 있습니다.

따라서 work 그룹을 사용하여 비공개 채널 기능을 구현할 수 있습니다. 그러나 서버와 바인딩된 비공개 채널에 대한 정보는 비즈니스측에 저장되어야 합니다.

채널 유형

Discord에서 채널을 생성할 때 채널 유형을 음성 또는 텍스트로 설정할 수 있습니다. 텍스트 채널은 텍스트, 이모티콘, 이미지를 기반으로 채팅이 가능하고 음성 채널은 음성/영상 채팅이 가능합니다. 사용자는 동시에 하나의 음성 채널에만 참여할 수 있습니다. 새로운 음성 채널에 참여하려면 사용자는 현재 참여하고 있는 음성 채널에서 나가야 합니다.

다음 4가지 사항에 주의하십시오.

1. 채널 생성 시 채널 유형을 설정해야 합니다. 채널 생성 섹션에서 설정 방법을 찾으실 수 있습니다.
2. 사용자가 음성 채널에 참여할 때 시스템은 사용자가 이미 다른 음성 채널에 있는지 확인해야 합니다.
3. 음성 채널은 전역적입니다.
4. Tencent Cloud IM은 현재 사용자 음성 채널 참여 여부와 어떤 음성 채널에 참여 중인지 쿼리하는 API를 제공하지 않습니다. 해당 관련 데이터는 비즈니스측에서 유지 관리해야 합니다.

상기 4번 사항과 관련하여 개발자는 Tencent Cloud IM에서 제공하는 그룹 참여 및 탈퇴 콜백을 사용하여 사용자의 음성 채널 상태를 유지하고 비즈니스측에 상태를 저장할 수 있습니다. Tencent Cloud IM에서 제공하는 콜백에는 지연이 있을 수 있습니다. 즉, 사용자가 한 음성 채널을 떠난 후 짧은 시간 동안 다른 음성 채널에 참여하지 못할 수 있습니다. 따라서 이 문제를 해결하기 위해 클라이언트를 사용하여 사용자의 음성 채널 참여 또는 퇴장 상태를 비즈니스 서버에 실시간으로 리포트할 수도 있습니다.

채널에 사용자 초대

사용자의 채널 참여는 다음 세 가지 방법이 있습니다.

1. 다른 사용자의 초대를 받아 서버에 가입합니다. 사용자가 서버에 가입하면 사용자는 서버의 모든 공개 채널에 가입됩니다.
2. 서버 검색을 통해 서버에 가입합니다.
3. 서버 관리자의 초대를 받아 비공개 채널에 가입합니다.

커뮤니티의 특성에 따라 사용자는 서버에 가입하기만 하면 공개 채널에 가입할 수 있습니다.

1. 정확한 그룹 ID 검색을 통한 서버 가입을 지원합니다.
2. 초대를 통한 서버 가입을 지원합니다.
3. 서버 가입 신청을 지원하며 승인이 필요하지 않습니다.

채널 설정

채널을 음소거 또는 음소거 해제하고 채널에 대한 알림을 구성할 수 있습니다. 해당 API는 각각 setAllMute 및 setGroupReceiveMessageOpt입니다.

```
// 기본 채널 정보 설정
V2TIMManager.getGroupManager().setTopicInfo(topicInfo, new V2TIMCallback() {
@Override
public void onSuccess() {
// 설정 완료
}

@Override
public void onError(int i, String s) {
// 설정 실패
}
});
```

```
// 해당 채널의 메시지 수신 옵션 설정
String groupID = "topicid"
int opt = 0;
V2TIMManager.getMessageManager().setGroupReceiveMessageOpt(groupID, opt, new V2TI
MCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int i, String s) {

}
});
```

채널 메시지 목록

getHistoryMessageList API를 사용하여 채널의 기록 메시지를 가져올 수 있습니다.

```
final V2TIMMessageListGetOption option = new V2TIMMessageListGetOption();
option.setGroupID("채널 ID");
option.setCount(20);
// 기타 구성
V2TIMManager.getMessageManager().getHistoryMessageList(option, new V2TIMValueCall
back<List<V2TIMMessage>>() {
@Override
public void onSuccess(List<V2TIMMessage> v2TIMMessages) {
// 채널 기록 메시지 가져오기 완료
```

```
}

@Override
public void onError(int code, String desc) {
// 채널 기록 메시지 가져오기 실패
}
});
```

채널의 읽지 않은 메시지 수

채널의 읽지 않은 메시지 수는 서버의 읽지 않은 메시지 수와 다릅니다. 서버의 읽지 않은 메시지 수는 대화 정보에 있고 채널의 읽지 않은 메시지 수는 채널 기본 정보에 있습니다. Tencent Cloud IM에서 제공하는 그룹 콜백 onTopicInfoChanged를 사용하여 읽지 않은 메시지 수를 실시간으로 가져올 수 있습니다.

```
String groupID = "서버 ID";
List< String > topicIDList= new LinkedList(); // 채널 메시지 목록
V2TIMManager.getGroupManager().getTopicInfoList(groupID, topicIDList, new V2TIMVa
lueCallback<List<V2TIMTopicInfoResult>>() {
@Override
public void onSuccess(List<V2TIMTopicInfoResult> v2TIMTopicInfoResults) {
// 채널 ID, 이름, 읽지 않은 메시지 수와 같은 채널 정보 가져오기
}

@Override
public void onError(int i, String s) {

}
});
```

채널 구성원 목록

서버에 가입하는 사용자는 기본적으로 서버 아래의 모든 공개 채널에 가입됩니다. 따라서 공개 채널 구성원 목록을 가져오려면 서버의 그룹 구성원 목록만 가져오면 됩니다. 다음은 공개 채널 구성원 목록을 가져오는 방법입니다.

```
String groupID = "서버 ID";
int filter = 0; // 그룹 구성원 역할: 관리자, 일반 구성원...
long nextSeq = 0;// 페이징 매개변수
V2TIMManager.getGroupManager().getGroupMemberList(groupID, filter, nextSeq , new
V2TIMValueCallback<V2TIMGroupMemberInfoResult>() {
@Override
public void onError(int i, String s) {
CommonUtil.returnError(result,i,s);
}

@Override
```

```
public void onSuccess(V2TIMGroupMemberInfoResult v2TIMGroupMemberInfoResult) {

}
});
```

비공개 채널의 구성원 목록을 가져오려면 getGroupMemberList API를 호출하여 비공개 채널의 그룹 ID를 전달하면 됩니다.

스레드

스레드는 채널의 특정 토픽에 대한 토론 그룹입니다. 사용자는 스레드 내의 특정 토픽에 대한 메시지를 찾아볼 수 있으며, 스레드 요약은 채널의 메시지 목록에서도 볼 수 있습니다. 스레드도 Tencent Cloud IM에서 그룹으로 간주됩니다.

스레드 생성

스레드는 별도로 생성하거나 채널의 메시지와 바인딩하여 생성할 수 있습니다. Tencent Cloud IM에서 제공하는 createGroup API를 사용하여 스레드를 생성할 수 있습니다. 다음 사항에 주의하십시오.

1. 스레드가 생성되면 기본적으로 현재 서버의 모든 구성원이 스레드에 있습니다.
2. 스레드가 생성된 후 서버에 가입한 구성원도 스레드에 추가됩니다.
3. 나중에 스레드에 참여하는 사용자는 스레드가 생성된 이후의 스레드 채팅 기록을 볼 수 있습니다.

상기 내용을 요약하면, 스레드/그룹이 생성되면 서버의 그룹 구성원이 스레드에 추가되고, 사용자가 서버에 참여하려면 사용자 그룹 참여 콜백에서 서버의 모든 스레드에 사용자를 추가해야 합니다. 다음 API를 사용하여 비즈니스 서버 측에서 이 두 단계를 구현하는 것이 가장 좋습니다.

1. Getting Group Member Profiles.
2. Creating a Group.
3. Adding Group Members.

서버 측 콜백은 After a User Joins a Group 콜백입니다.

스레드 수

채널 목록에서 채널의 스레드 목록을 가져올 수 있습니다. 따라서 스레드와 서버 간에 다대일 관계를 설정해야 합니다. 스레드에 기록 메시지가 있는 경우 스레드와 메시지 간의 일대일 관계도 설정해야 합니다. Tencent Cloud IM은 현재 이러한 관계를 설정할 수 있는 기능을 제공하지 않으므로, 비즈니스측에서 이러한 관계를 유지 관리해야 합니다. 비즈니스측에서 제공하는 HTTP API를 사용하여 스레드 수와 스레드 목록을 가져오고, 온라인 메시지 전송을 통해 스레드 목록을 실시간으로 업데이트할 수 있습니다.

스레드 요약

사용자가 메시지에 대한 스레드를 생성하면 사용자는 메시지 목록에서 다음과 같은 메시지 요약 정보를 볼 수 있습니다.

1. 스레드 메시지 수.
2. 스레드 lastMessage 관련 정보.
3. 기타 실시간으로 메시지 목록에 표시되어야 하는 정보

스레드 메시지 요약 기능을 구현하려면 그룹 메시지 전송 콜백 및 메시지 편집 기능을 사용해야 합니다. 사용자가 스레드에서 메시지를 보낸 후, IM 서비스는 메시지 전송 콜백을 트리거하고 관련 정보를 비즈니스측에 동기화합니다. 그 다음 비즈니스는 스레드의 메시지 수를 세고, lastMessage 정보를 유지하는 동시에 메시지 편집 API를 통해 정보를 메시지에 저장합니다.

메시지 관련

메시지 피드백

메시지 피드백은 아래 이미지와 같이 사용자를 위한 메시지의 확장입니다.

모든 유형의 메시지는 편집 및 피드백이 지원되며, 커뮤니티 그룹 지원이 필요하므로 cloudCustomData 필드에 데이터를 저장하는 것이 좋습니다. 다음은 자세한 데이터 저장 형식(참고용)입니다.

```
{
"reaction": {
"simle":["user1","user2"]
}
}
```

```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new V2TIMComplete
Callback<V2TIMMessage>() {
@Override
public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
// 메시지 수정 완료
}
});
```

메시지 편집

메시지 편집은 메시지 피드백과 동일하게 작동하며, 구성된 사용자 지정 데이터의 몇 가지 차이점만 있습니다. 모든 메시지의 편집을 지원하려면 cloudCustomData 필드에서 데이터를 편집하는 것이 좋습니다. 데이터 형식은 다음과 같습니다.

```
{
"isEdited": true
}
```

```
V2TIMManager.getMessageManager().modifyMessage(modifiedMessage, new V2TIMComplete
Callback<V2TIMMessage>() {
@Override
public void onComplete(int i, String s, V2TIMMessage v2TIMMessage) {
// 메시지 편집 완료
}
});
```

메시지 별표

메시지 별표는 다른 사용자가 별표 표시된 메시지를 볼 수 있도록 그룹 채팅에서 메시지에 별표를 표시하는 것입니다. 커뮤니티 그룹은 그룹 속성을 지원하지 않으므로 사용자 지정 메시지를 사용하여 메시지 별표 기능을 구현합니다.

사용자 지정 그룹 메시지를 사용하려면 먼저 Tencent Cloud IM 콘솔에서 아래 이미지와 같이 구성해야 합니다.

그 다음 다음과 같이 설정합니다.

```
V2TIMGroupInfo info = new V2TIMGroupInfo();
info.setCustomInfo("pin data");
V2TIMManager.getGroupManager().setGroupInfo(info, new V2TIMCallback() {
@Override
public void onError(int i, String s) {
// 설정 실패
}

@Override
public void onSuccess() {
// 설정 성공
}
});
```

설정 성공 후 그룹 구성원은 onMemberInfoChanged 이벤트를 수신할 수 있으며 오프라인 사용자도 그룹 프로필을 통해 별표 표시된 메시지 콘텐츠를 얻을 수 있습니다.

```
List< String > groupIDList = new LinkedList();
V2TIMManager.getGroupManager().getGroupsInfo(groupIDList, new V2TIMValueCallback<
List<V2TIMGroupInfoResult>>() {
@Override
public void onError(int i, String s) {

}

@Override
public void onSuccess(List<V2TIMGroupInfoResult> v2TIMGroupInfoResults) {
```

```
}
});
```

사용자 지정 필드는 현재 채널이 아닌 서버에서만 구성할 수 있습니다.

입력 중 상태

친구가 입력 중일 때 다른 클라이언트의 사용자는 상기 이미지와 같이 사용자의 입력 중 상태를 볼 수 있습니다. Tencent Cloud IM의 온라인 메시지 기능을 사용하여 이 기능을 구현할 수 있습니다. 입력 중 상태 메시지는:

1. 온라인 사용자만 받을 수 있습니다.
2. 로밍 서버에 저장되지 않습니다.

또한 성능 향상을 위해 다음과 같이 최적화되었습니다.

- 입력 중 상태 메시지 전송은 두 사용자가 20s 이내에 서로 메시지를 보낸 경우에만 트리거됩니다.
- 동일한 문자 메시지는 온라인 메시지 전송을 여러 번 트리거하지 않습니다.

비공개 메시지

비공개 메시지는 Discord 사용자 간 전송하는 메시지이며, 친구 여부는 관계가 없습니다.

- 친구가 아닌 다른 사용자에게 비공개 메시지를 보내려면 해당 사용자의 사용자 ID를 알아야 하며, Tencent Cloud IM 콘솔에서 메시지 전송 점검을 비활성화해야 합니다. 그렇지 않으면 메시지 전송에 실패합니다.
- 또는 addFriend API를 호출하여 해당 사용자를 친구로 추가하고 getFriendList API를 사용하여 연락처를 가져올 수 있습니다.

관련 코드는 다음과 같습니다.

```
// 친구 추가
V2TIMManager.getFriendshipManager().addFriend(info, new V2TIMValueCallback<V2TIMF
riendOperationResult>() {
@Override
public void onError(int i, String s) {
// 친구 추가 실패
}

@Override
public void onSuccess(V2TIMFriendOperationResult v2TIMFriendOperationResult) {
// 친구 추가 성공
}
});
```

```
// 연락처 가져오기
V2TIMManager.getFriendshipManager().getFriendList(new V2TIMValueCallback<List<V2T
IMFriendInfo>>() {
@Override
public void onError(int i, String s) {
// 연락처 가져오기 실패
}

@Override
public void onSuccess(List<V2TIMFriendInfo> v2TIMFriendInfos) {
// 연락처 가져오기 성공
}
});
```

마이페이지

온라인 상태

Discord 사용자 패널의 온라인 상태 기능은 Tencent Cloud IM에서 제공하는 온라인 상태 API로 다음과 같이 구현할 수 있습니다.

- setSelfStatus를 통해 자신만의 사용자 지정 온라인 상태를 설정하기 위한 API입니다. 사용자의 온라인/오프라인 상태는 Tencent Cloud IM에서 설정하며 개발자가 수정할 수 없습니다.
- getUserStatus를 통해 친구의 온라인 상태를 가져옵니다.
- onUserStatusChanged를 통해 친구의 온라인 상태 변경을 수신합니다.

관련 코드는 다음과 같습니다.

```
// 자신의 온라인 상태 설정
V2TIMUserStatus customStatus = new V2TIMUserStatus();
V2TIMManager.getInstance().setSelfStatus(customStatus, new V2TIMCallback() {
@Override
public void onSuccess() {

}

@Override
public void onError(int i, String s) {

}
});

// 친구의 온라인 상태 가져오기
List<String> userIDList = new LinkerList();
```

```
V2TIMManager.getInstance().getUserStatus(userIDList, new V2TIMValueCallback<List<
V2TIMUserStatus>>() {
@Override
public void onSuccess(List<V2TIMUserStatus> v2TIMUserStatuses) {

}

@Override
public void onError(int i, String s) {

}
});
```

개인 정보 관련

개인 정보 관련 기능은 Tencent Cloud IM에서 제공하는 관계 체인 관련 API로 다음과 같이 구현할 수 있습니다.

- 사용자 프로필 가져오기 getUsersInfo.
- 사용자 프로필 설정 setSelfInfo.

```
// 사용자 프로필 설정
final V2TIMUserFullInfo userFullInfo = new V2TIMUserFullInfo();
V2TIMManager.getInstance().setSelfInfo(userFullInfo, new V2TIMCallback() {
@Override
public void onError(int i, String s) {

}

@Override
public void onSuccess() {

}
});
```

```
// 사용자 프로필 가져오기
List<String> userIDList = new LinkedList();
V2TIMManager.getInstance().getUsersInfo(userIDList, new V2TIMValueCallback<List<V
2TIMUserFullInfo>>() {
@Override
public void onError(int i, String s) {

}

@Override
public void onSuccess(List<V2TIMUserFullInfo> v2TIMUserFullInfos) {
```

```
}
});
```

기타

검색 기능

Discord는 다음과 같은 검색 기능을 제공합니다.

Tencent Cloud IM은 다음과 같은 풍부한 검색 기능을 제공합니다.

- 메시지 검색 searchLocalMessages.
- 그룹 검색 searchGroups.
- 그룹 구성원 검색 searchGroupMembers.
- 친구 검색 searchFriends.

API 사용 방법은 공식 문서를 참고하십시오.

오프라인 푸시

Tencent Cloud IM의 온라인 메시지는 오프라인 사용자에게 도달할 수 없습니다. 이 문제를 해결하려면 디바이스 벤더가 제공하는 오프라인 푸시 기능을 Tencent Cloud IM에 통합해야 합니다. 자세한 내용은 오프라인 푸시를 참고하십시오.

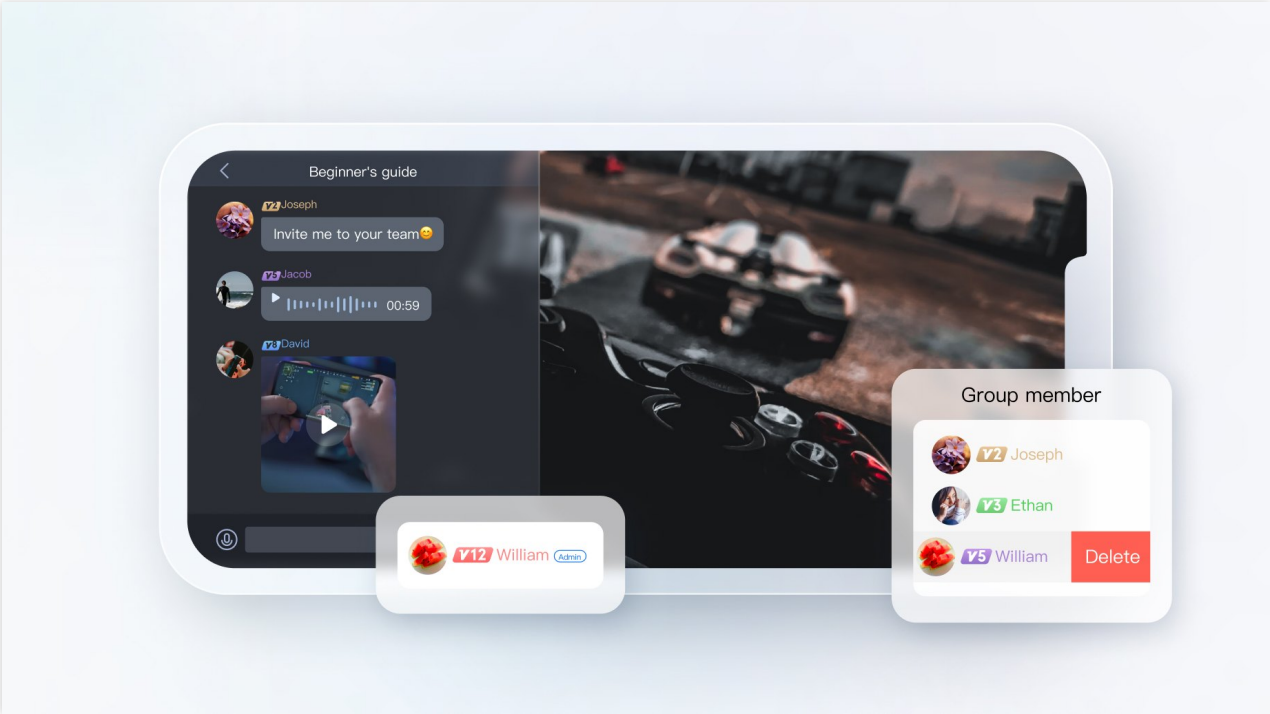
민감한 문자 확인

Discord에서 정보 및 구성을 보낼 때 콘텐츠 필터링이 필요합니다. Tencent Cloud IM은 또한 사용자가 규정을 준수하여 IM을 사용할 수 있도록 이러한 체계를 제공합니다.

IM을 게임에 통합하는 방법

최종 업데이트 날짜: : 2024-02-22 14:20:12

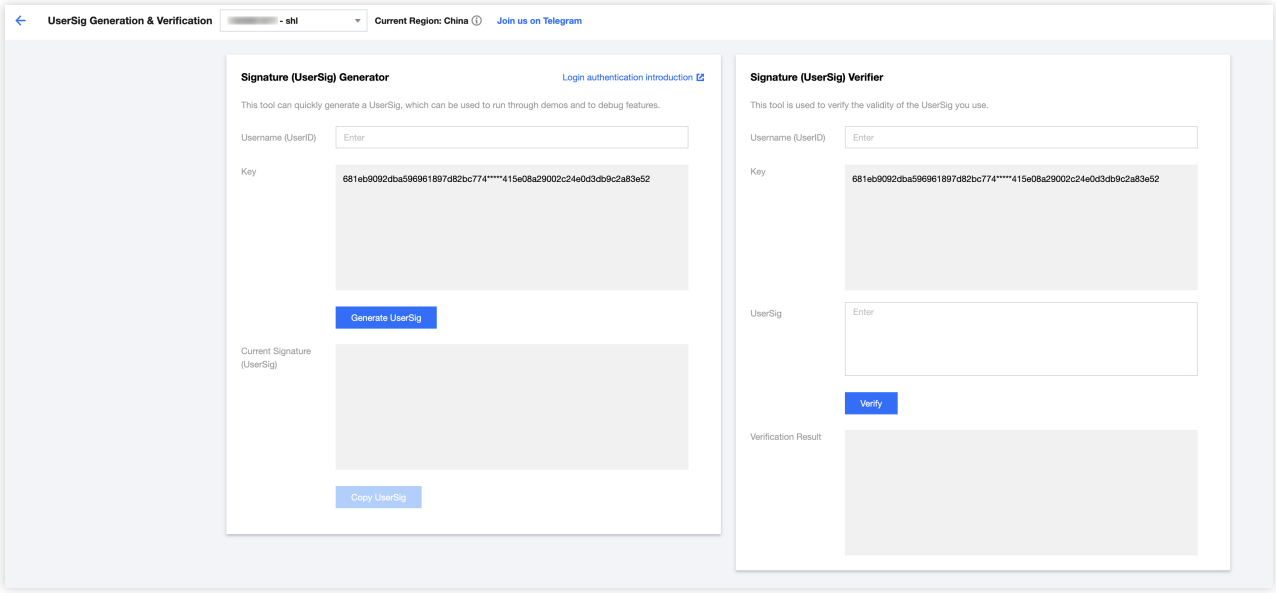
인스턴트 메시징은 게임의 일반적인 요구 사항이며 인스턴트 채팅은 멀티플레이어 게임의 필수 기능이 되었습니다. 게임 플랫폼 자체에는 다양한 그룹 유형, 사용자 지정 메시지 유형(예: 게임 내 소품 선물 및 거래), 전역 액세스 및 기타 복잡한 요구 사항이 포함됩니다. 이 문서는 게임 내 채팅을 구축하는 과정에서 일반적인 요구 사항 구현 방법과 가능한 문제 및 고려 사항을 정리하여 개발자가 비즈니스를 빠르게 이해하고 요구 사항을 구현하도록 돕습니다.



준비 작업

키로 UserSig 계산하기

IM 계정 시스템에서 사용자 로그인에 필요한 암호는 IM에서 제공한 키를 사용하여 서버에서 계산합니다. 자세한 내용은 Generating UserSig을 참고하십시오. 개발 단계에서 클라이언트의 개발 지연을 방지하기 위해 아래와 같이 콘솔에서 UserSig 계산할 수도 있습니다.



관리자 계정 구성

게임 내 인스턴트 메시징을 관리하려면 관리자가 게임에 이메일 알림을 보내고 임시 팀 구성 메시지를 관리하는 등 IM 서버 API를 통해 수행할 수 있습니다. 이러한 API를 호출하려면 IM 관리자 계정 생성이 필요합니다. 기본적으로 IM은 관리자의 사용자 ID가 있는 계정을 제공합니다. 필요에 따라 여러 관리자 계정을 만들 수도 있습니다. 최대 5개의 관리자 계정을 만들 수 있습니다.

콜백 주소 구성 및 콜백 활성화

게임 내 팀 구성 및 기타 요구 사항을 구현하려면 특정 시나리오에서 IM 백엔드가 비즈니스 백엔드를 호출하는 콜백 이벤트를 사용합니다. 아래와 같이 HTTP API를 제공하고 콘솔 > 콜백 구성 모듈에서 구성하기만 하면 됩니다.

클라이언트 SDK 통합

준비를 완료한 후 IM 클라이언트 SDK를 프로젝트에 통합해야 합니다. 필요에 따라 다른 통합 옵션을 선택할 수 있습니다. 자세한 지침은 TUIKit 소개를 참고하십시오.
다음은 게임에 통합될 일반적인 IM 기능을 설명하고 구현 코드와 함께 모범 사례를 제공합니다.

게임 채팅방 기능 개발 가이드

사용자 프로필

일반 사용자 프로필

게임 사업에 저장된 일반 사용자 프로필은 기본 정보 프로필과 기타 정보 프로필로 나눌 수 있습니다.

기본 정보	기타 정보
아이디, 성별, 생년월일, 등급, 역할, 휴대폰 번호 등	게임에 필요한 기타 정보

프로필 저장

게임 비즈니스는 사용자가 많고 방대한 양의 사용자 데이터를 저장하기 어렵습니다. Tencent Cloud IM은 사용자 프로필 호스팅 기능을 통해 완벽한 프로필 솔루션 세트를 제공합니다. 다음은 사용자 프로필을 Tencent Cloud IM에 저장하는 것과 비즈니스 백그라운드에 저장하는 것을 비교한 것입니다.

아이템	IM	비즈니스 백그라운드
저장 용량	자동 스케일링 지원	제한되고 확장하기 어려운 용량
사용자 프로필	필드의 길이와 이름에 대한 제한이 있는 표준 및 사용자 정의 필드 제공	사용자 정의 가능, 더 유연함
프로필 읽기/쓰기	사용하기 쉬운 서비스 API 및 가이드라인 제공	자체 개발 필요
API	API 호출 빈도: 초당 200회 이하	필요에 따라 API 호출 및 기타 기능 개발 가능
보안성	원격 재해 복구 및 교차 리전 배포 지원	자체 유지 보수 필요

프로필 저장 및 읽기/쓰기 기능 외에도 IM에는 사용자 프로필 호스팅 서비스 덕분에 다음과 같은 이점이 있습니다.
1. IM은 원격 재해 복구, 리전 간 배포 및 오토스케일링 기능을 제공합니다. 이러한 방식으로 서버 다운타임, 다중 복사본 1차-2차 복제, 용량 스케일링과 같은 복잡한 처리 흐름을 손쉽게 해결할 수 있습니다.
2. IM은 업계에서 일반적으로 사용되는 비즈니스 처리 흐름을 제공하므로 사용자 프로필 비즈니스 로직에 대해 걱정할 필요가 없습니다.

3. IM은 전문적인 운영 프로세스와 팀을 제공하여 매년 99.99%의 서비스 품질을 보장하고 안정적인 서비스를 제공할 수 있도록 도와드립니다.
4. IM은 사용하기 쉬운 서비스 API와 액세스하기 쉬운 가이드라인을 제공하며 전체 프로세스에 걸쳐 프리미엄 서비스를 제공합니다.

IM 사용자 프로필 저장 방법

IM 저장 스키마에는 사용자 프로필 저장 및 읽기/쓰기 기능이 포함됩니다. 다음은 IM이 사용자 프로필, 친구 프로필 및 확장 데이터를 저장하는 방법을 설명합니다. 모든 데이터는 `Key-Value` 형식으로 저장됩니다. `Key` 는 `String` 유형이며 영어 대문자와 소문자, 숫자 및 언더바만 지원합니다. `Value` 는 다음 유형을 지원합니다.

유형	설명
uint64_t	정수(사용자 정의 필드에서 지원되지 않음)
string	문자열. 길이 500바이트 이하
bytes	buffer. 길이 500바이트 이하
string 배열	문자열 배열. 각 문자열은 500바이트를 초과할 수 없으며 친구 목록의 Tag_SNS_IM_Group 필드에서만 사용 가능

사용자 프로필: 사용자 프로필에는 표준 및 사용자 정의 필드가 포함됩니다. 사용자 정의 필드의 경우 아래 확장 데이터 부분을 참고하십시오. 현재 IM에서 지원하는 표준 필드는 표준 프로필 정보 필드를 참고하십시오.
친구 프로필: 친구 프로필에는 표준 및 사용자 지정 친구 필드가 포함됩니다. IM의 연락처 목록은 최대 3000명의 친구를 지원합니다. 표준 친구 필드는 다음과 같이 설명됩니다.

필드 이름	유형	설명
Tag_SNS_IM_Group	Array	친구 목록
Tag_SNS_IM_Remark	string	친구 비고
Tag_SNS_IM_AddSource	string	친구 신청 출처
Tag_SNS_IM_AddWording	string	친구 신청 내용
Tag_SNS_IM_AddTime	Integer	친구 요청의 타임스탬프

자세한 내용은
표준 친구 필드
를 참고하십시오.
사용자 지정 프로필: 사용자 지정 프로필 필드를 신청하려면 앱 관리자가 IM 콘솔에 로그인하여 > 애플리케이션 구성 > 기능 구성으로 이동할 수 있습니다. 신청을 제출한 후 5분 후에 사용자 지정 프로필 필드가 적용됩니다.

사용자 프로필 관리에 대한 자세한 내용은 프로필 관리를 참고하십시오.
관계 체인 사용자 정의 프로필 관리에 대한 자세한 내용은 사용자 정의 친구 필드를 참고하십시오.

IM 사용자 프로필 저장 제한

서비스 기능 제한

데이터 저장: 각 string 또는 buffer는 500바이트를 초과할 수 없습니다.
사용자 정의 필드: 사용자 정의 필드의 키워드는 길이가 8바이트 이하인 영문자로 구성되어야 합니다. 사용자 정의 필드 값은 500바이트를 초과할 수 없습니다.
친구 관계망: 각 사용자는 최대 3000명의 친구를 가질 수 있습니다.

API 관련 제한

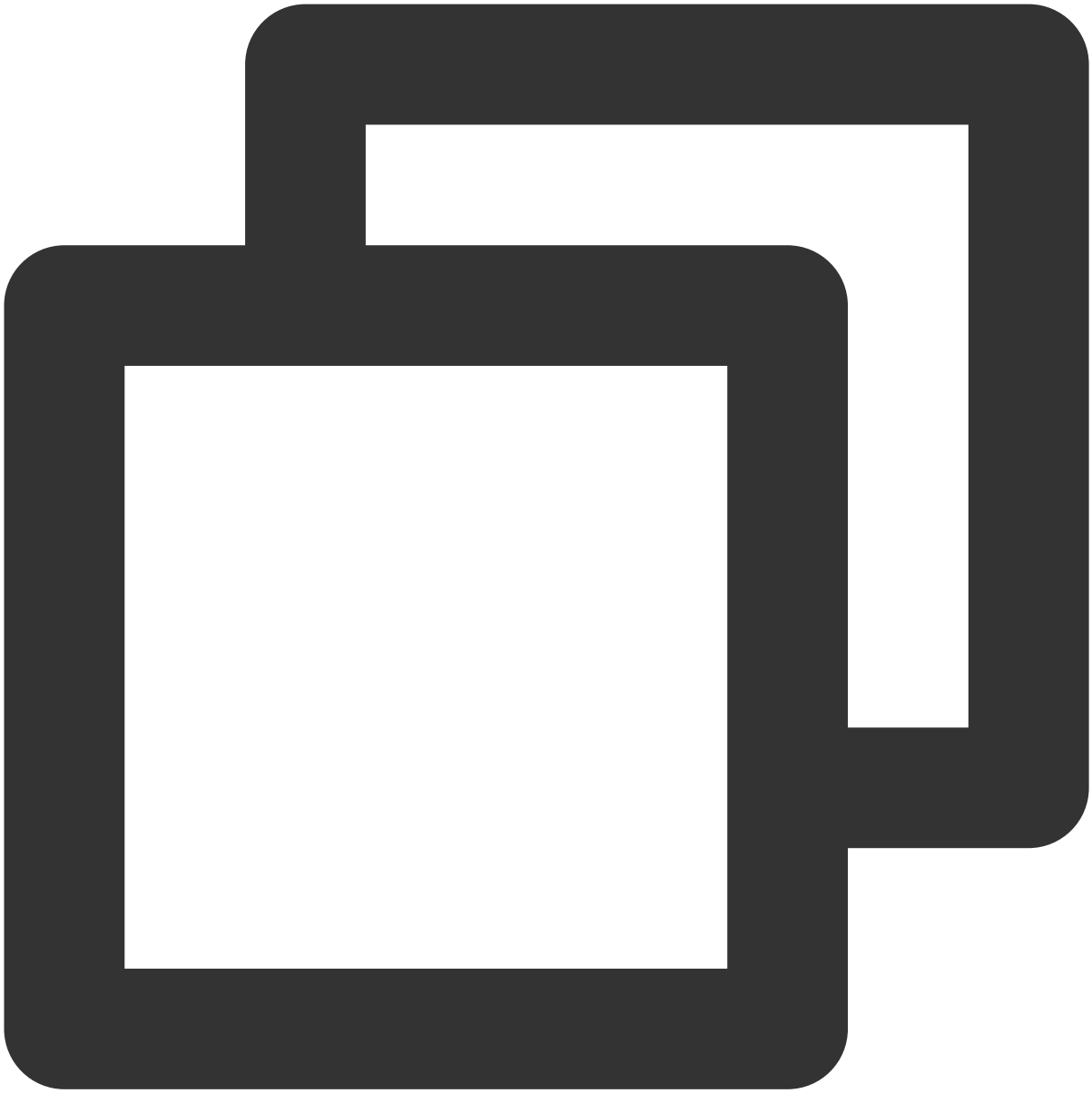
계정 관리: 한 번에 최대 100개의 사용자 이름을 가져올 수 있으며 요청당 최대 500명의 사용자 상태를 쿼리할 수 있습니다.
기타 호출 빈도: 초당 최대 200회
사용 제한에 대한 자세한 내용은 사용 제한을 참고하십시오.

이메일 시스템

이메일 시스템은 요즘 게임에서 거의 필수입니다. 이메일에는 문자 메시지뿐 아니라 게임 소품 및 보상과 같은 이메일 첨부 파일이 포함될 수 있습니다. 단일 사용자에게 이메일을 보내거나 이벤트 보상을 제공하기 위해 그룹 이메일로 보낼 수 있습니다. 다음은 이메일 수신 플레이어, 이메일 목록, 읽지 않은 이메일 수, 모든 회원에게 이메일 보내기, 이메일 유효 기간 등 다양한 측면에서 이메일 시스템의 기능을 구현하는 방법을 설명합니다.

이메일 수신 및 전송

플레이어 이메일 수신: 시스템 이메일이 성공적으로 전송되고 플레이어가 온라인 상태일 때 플레이어는 시스템 이메일을 제대로 수신할 수 있습니다. 플레이어는 이메일 대화의 메시지 목록을 가져와 기록 또는 최신 이메일을 얻을 수 있습니다. 또한 모든 유형의 새 메시지(텍스트, 사용자 지정 및 리치 미디어 메시지 포함)를 수신하기 위해 리스너를 추가하거나 삭제할 수 있습니다. Unity의 샘플 코드는 다음과 같습니다.



```
// 메시지 수신 이벤트 리스너 구성
TencentIMSDK.AddRecvNewMsgCallback((List<Message> messages, string user_data)=>{
foreach(Message message in messages)
{
  foreach (Elem elem in message.message_elem_array)
  {
    // 다음 요소가 있습니다.
    if (elem.elem_type == TIMElemType.kTIMElem_Text)
    {
       string text = elem.text_elem_content;
    }
```

```
    }
  }
  })
  // `RecvNewMsgCallback` 콜백을 수신하여 메시지 수신
  // 메시지 수신을 중지하려면 `RemoveRecvNewMsgCallback`을 호출하여 리스너를 제거합니다. 이 단
```

자세한 내용은 Unity-Receiving a Message를 참고하십시오.

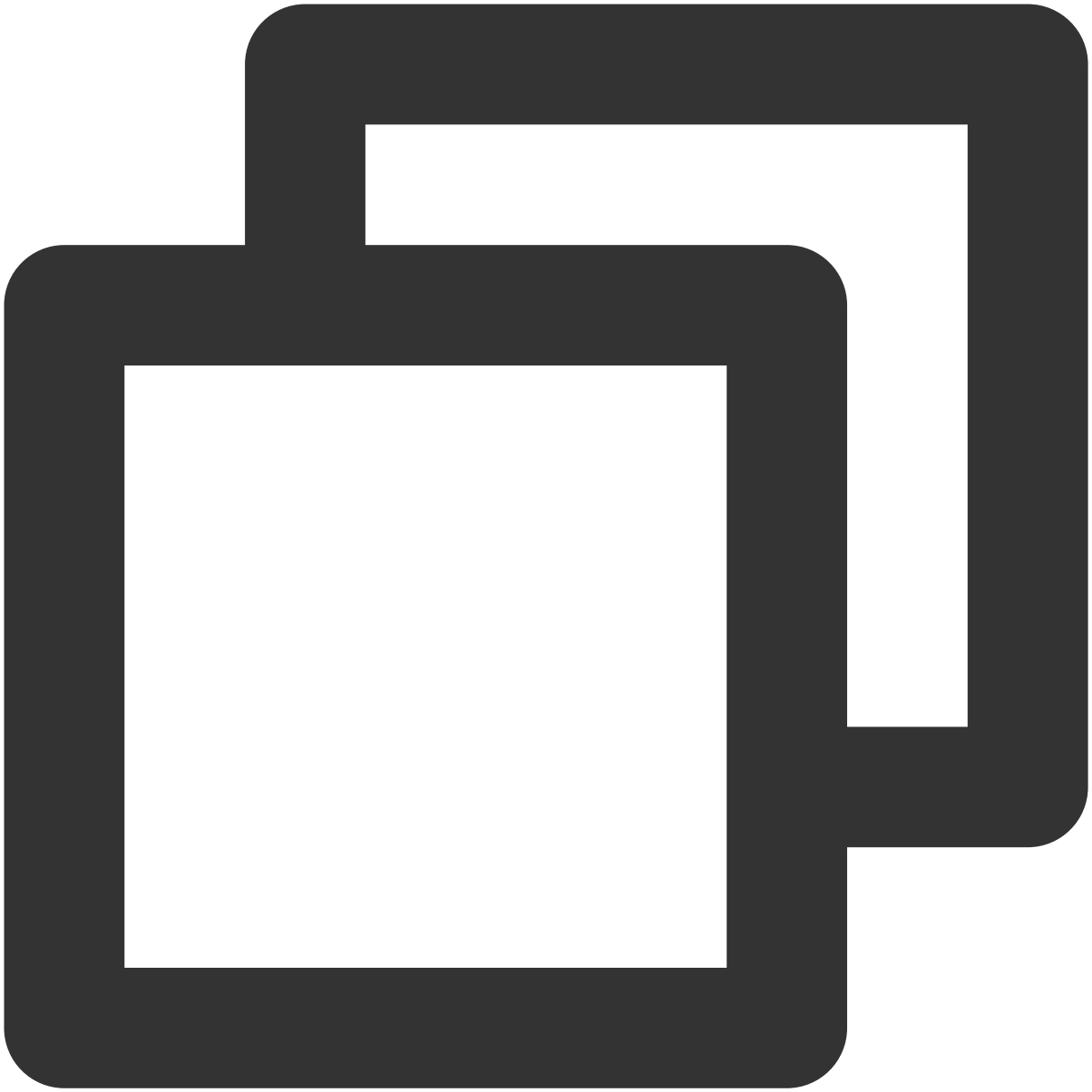
시스템 이메일 전송: 시스템은 다음과 같이 설명된 다른 서버 API를 통해 사용자에게 시스템 이메일을 보낼 수 있습니다.

API	특성	응용 시나리오
Sending One-to-One Messages to One User	지정된 계정으로 메시지를 보냅니다. 수신자에게 표시되는 발신자는 관리자가 아니라 관리자가 지정한 계정입니다.	특정 사용자에게 메시지 전송(예: 사용자에게 게임 보상 메시지 전송)
Sending One-to-One Messages to Multiple Users	일대일 메시지는 한 번에 최대 500명의 사용자에게 보낼 수 있으며 최대 통화 빈도는 초당 200회입니다.	수신자 그룹을 만들지 않고도 특정 사용자에게 메시지를 보낼 수 있습니다. 받는 사람이 많으면 메시지를 일괄적으로 보내야 합니다.
Sending Ordinary Messages in a Group	그룹에 일반 메시지를 보낼 때 모든 수신자를 동일한 그룹에 추가해야 합니다.	다수의 사용자에게 일반 메시지 전송 (community 그룹당 사용자 최대 10만명)
API for Pushing to All Users	App의 모든 사용자에게 메시지를 푸시합니다. 메시지 전송을 위한 사용자 태그 및 속성을 지정할 수 있습니다.	App의 모든 사용자 또는 캠페인 이메일과 같은 특정 특성 속성을 가진 다수의 사용자에게 메시지를 푸시합니다.

설명：

'모든 사용자에게 푸시'는 플래그십 버전에서만 사용할 수 있습니다. 이를 사용하려면 플래그십 버전 구매 후 콘솔 > 기능 구성 > 로그인 및 메시지 > 모든 사용자에게 푸시를 선택하고 기능을 활성화해야 합니다.

다음은 그룹에서 일반 메시지를 보내기 위한 기본 요청의 예시입니다.



```
{
    "GroupId": "@TGS#2C5SZEAEF",
    "Random": 8912345, // 임의의 숫자입니다. 두 메시지의 임의의 숫자가 5분 이내에 동일하면 동
    "MsgBody": [ // element 배열로 구성된 메시지 본문입니다. 자세한 내용은 필드 설명을 참고하
        {
            "MsgType": "TIMTextElem", // 텍스트
            "MsgContent": {
            "Text": "red packet"
            }
        },
        {
```

```
            "MsgType": "TIMCustomElem", // 사용자 지정
            "MsgContent": {
            "Data": "message",
            "Desc": "notification",
            "Ext": "url",
            "Sound":"dingdong.aiff"
            }
        }
    ],
}
```

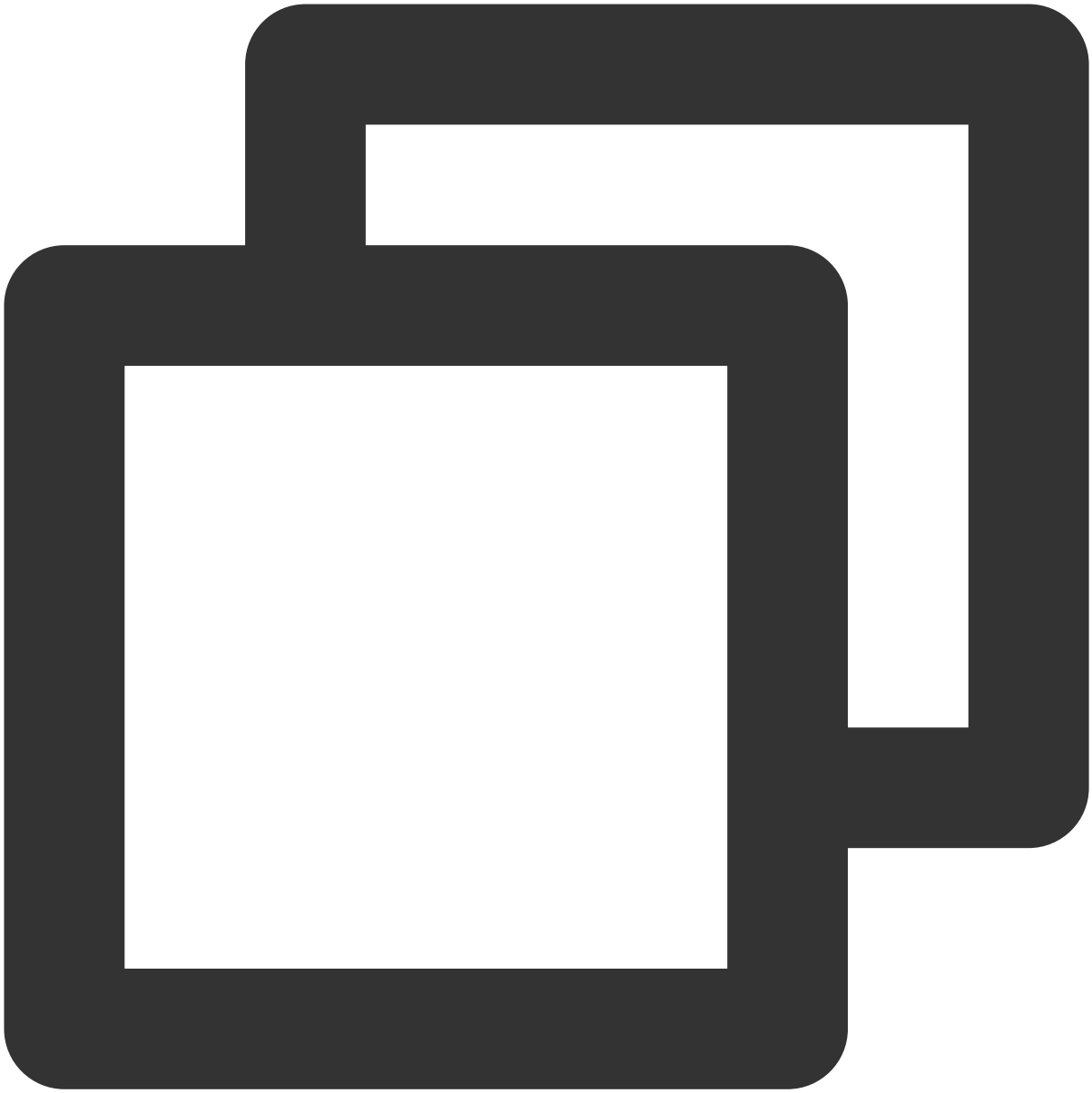
`MsgBody` (메시지 본문)는 메시지 배열입니다. 보낼 텍스트 및 사용자 지정 메시지를 추가할 수 있습니다.

이메일 목록

이메일 목록 기록 저장은 메시지 저장과 동일합니다. `C2C 메시지 기록 저장` 과 `그룹 메시지 기록 저장` 으로 구성됩니다. 그룹 채팅에는 최소 두 명의 사용자가 필요하므로 관리자가 지정한 계정과 저장을 위해 이메일을 받는 사용자를 포함하는 그룹을 만들 수 있습니다.

설명：

무료 버전 및 프로 버전의 경우 저장 기간은 7일입니다. 플래그십 버전의 경우 저장 기간은 30일입니다. 프로 버전 및 플래그십 버전을 사용하면 저장 기간을 연장할 수 있습니다. 콘솔에 로그인하여 관련 구성을 수정할 수 있습니다.
메시지 기록 저장 기간을 연장하는 것은 유료 부가 서비스입니다. 청구에 대한 자세한 내용은 부가 서비스 요금을 참고하십시오.
네트워크가 정상이면 최신 클라우드 데이터를 가져옵니다. 비정상적인 경우 SDK는 로컬에 저장된 기록 메시지를 반환합니다. 페이징 풀링이 지원됩니다. 다음은 이전 이메일 목록을 풀링하기 위한 Unity 예시 코드입니다.

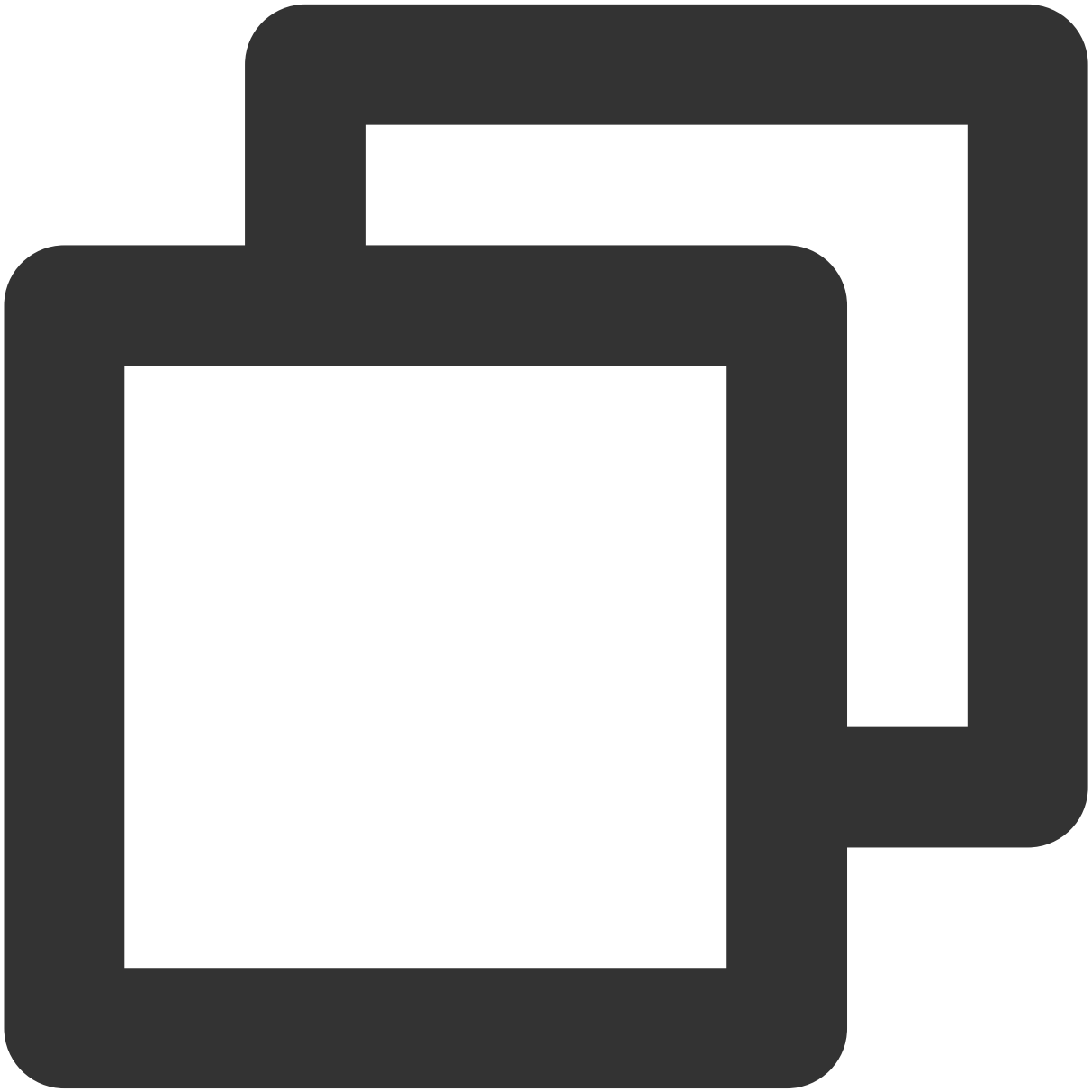


```
// 일대일 메시지 기록 풀링
// 첫 번째 풀링에 대해 msg_getmsglist_param_last_msg를 null로 설정
// msg_getmsglist_param_last_msg는 두 번째 풀링에 대해 반환된 메시지 목록의 마지막 메시지일
var get_message_list_param = new MsgGetMsgListParam
{
    msg_getmsglist_param_last_msg = LastMessage
};
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
// 콜백 로직 프로세스
});
```

메시지 기록을 가져오는 방법에 대한 자세한 내용은 Historical Messages - Unity를 참고하십시오.

읽지 않은 이메일 수

사용자-시스템 이메일의 기록은 채팅의 대화와 같습니다. Tencent Cloud IM은 사용자가 아직 메시지를 읽지 않았음을 사용자에게 상기시키는 읽지 않은 대화 개수 기능을 제공합니다. 사용자가 대화를 클릭하고 대화 목록으로 돌아가면 읽지 않은 메시지 수가 지워집니다. 다음은 Unity 예시 코드입니다.



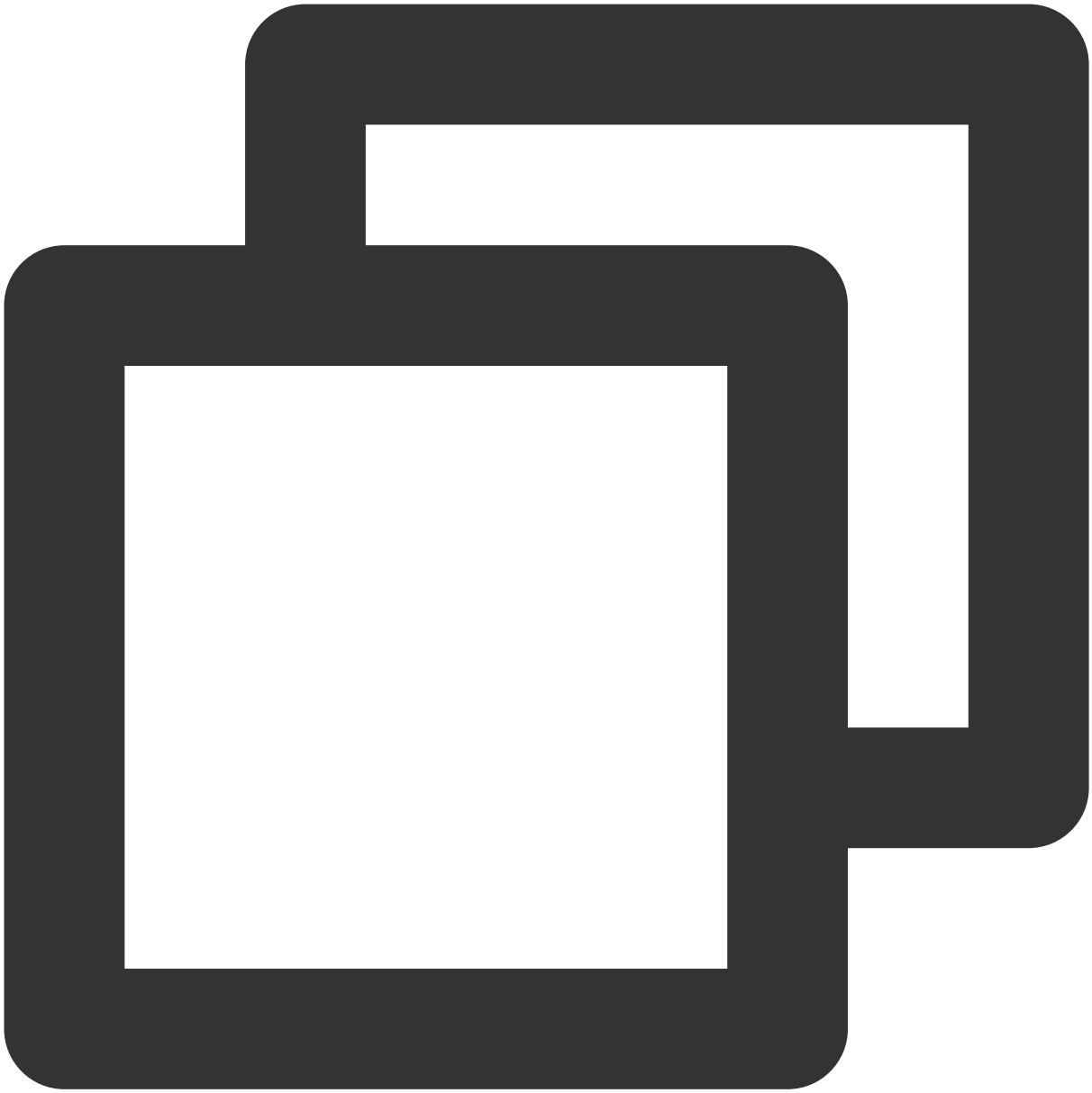
```
// 읽지 않은 총 개수 가져오기
TIMResult res = TencentIMSDK.ConvGetTotalUnreadMessageCount((int code, string desc,
 // 비동기 로직 프로세스
```

```
});

// 읽지 않은 개수 변경 알림
TencentIMSDK.SetConvTotalUnreadMessageCountChangedCallback((int total_unread_count,
 // 콜백 로직 프로세스
});

// 모든 대화의 읽지 않은 수 지우기
TIMResult res = TencentIMSDK.MsgMarkAllMessageAsRead((int code, string desc, string
 // 비동기 로직 프로세스
});
```

자세한 내용은 Unity - Total Unread Count of All the Conversations를 참고하십시오.

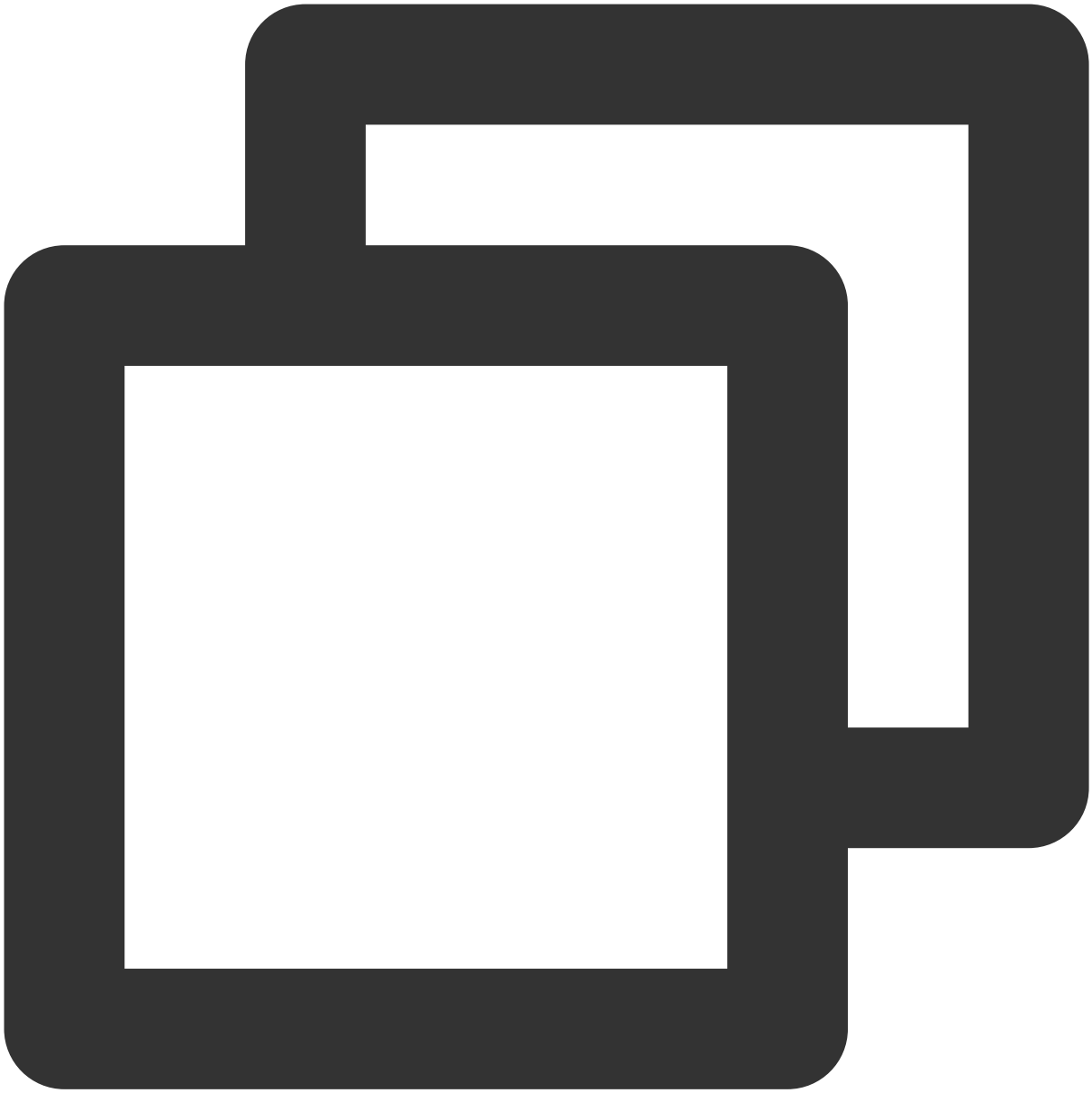
모든 구성원에게 이메일 보내기

모든 구성원에게 이메일 보내기는 게임의 모든 플레이어에게 이메일 메시지를 보내는 것입니다. Tencent Cloud IM은 서버 측에서 `모든 구성원에게 푸시` 기능을 제공합니다. 다음은 예시 코드입니다.



```
https://console.tim.qq.com/v4/all_member_push/im_push?usersig=xxx&identifier=admin&
```

다음은 예시 요청입니다.



```
{
    "From_Account": "admin",
    "MsgRandom": 56512,
    "MsgLifeTime": 120, // 120초(2분) 동안 오프라인 저장
    "MsgBody": [
        {
        "MsgType": "TIMTextElem",
        "MsgContent": {
            "Text": "hi, beauty"
            }
        }
```

```
        ]
    }
```

모든 구성원에게 메시지를 보내는 기능을 사용하면 일부 사용자가 온라인 상태가 아니더라도 지정된 오프라인 저장 기간 내에 메시지를 받을 수 있도록 오프라인 저장 기간을 설정할 수 있습니다. `MsgLifeTime` (단위: 초)을 구성하여 오프라인 저장 기간을 지정합니다. 최대 기간은 604800s(7일)입니다. 기본값은 0이며 메시지가 오프라인으로 저장되지 않음을 나타냅니다.
모든 사용자에게 푸시에 대한 자세한 내용은 Pushing to All Users를 참고하십시오.

이메일 유효 기간

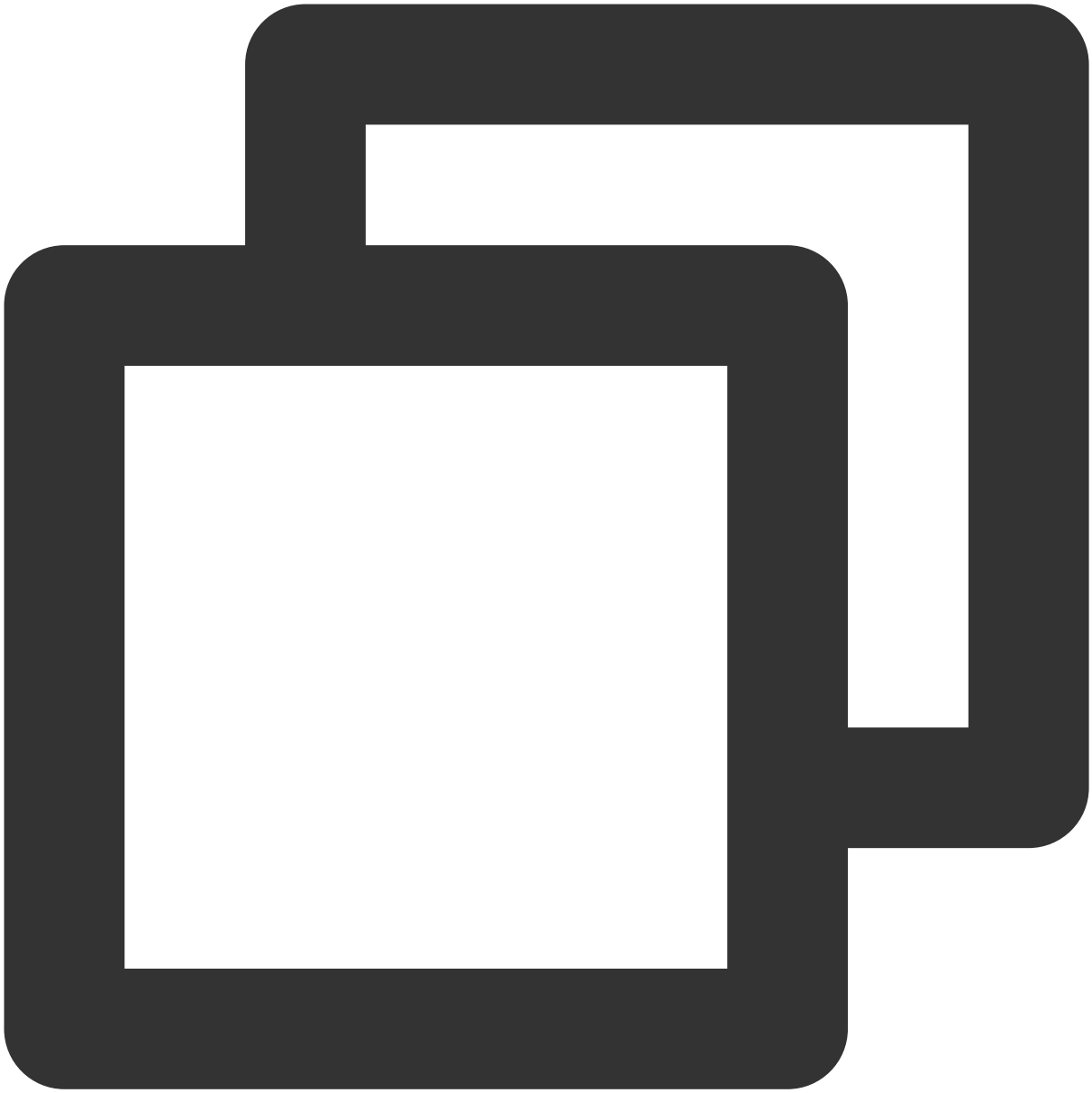
기록 이메일의 저장 기간은 다음과 같습니다. 무료 버전 및 프로 버전의 경우 저장 기간은 7일입니다. 플래그십 버전의 경우 보관 기간은 30일입니다. 프로 버전 및 플래그십 버전을 사용하면 저장 기간을 연장할 수 있습니다. 기록 이메일 저장에 대한 자세한 내용은 메시지 기록 저장 기간 설정을 참고하십시오.
또한 서버 측에서 메시지를 보낼 때 `MsgLifeTime` 을 설정하여 메시지의 오프라인 저장 기간(최대 7일)을 지정할 수 있습니다. `MsgLifeTime` 이 0이면 메시지가 온라인 사용자에게만 전송되고 오프라인에 저장되지 않음을 나타냅니다. 자세한 내용은 서버 API를 참고하십시오.

임시 팀 구성

일시적인 팀 구성은 온라인 멀티플레이어 게임에서 필수적입니다. 다음은 임시 팀 구성 시나리오와 배경 및 팀원이 팀 정보를 얻는 방법을 설명합니다.

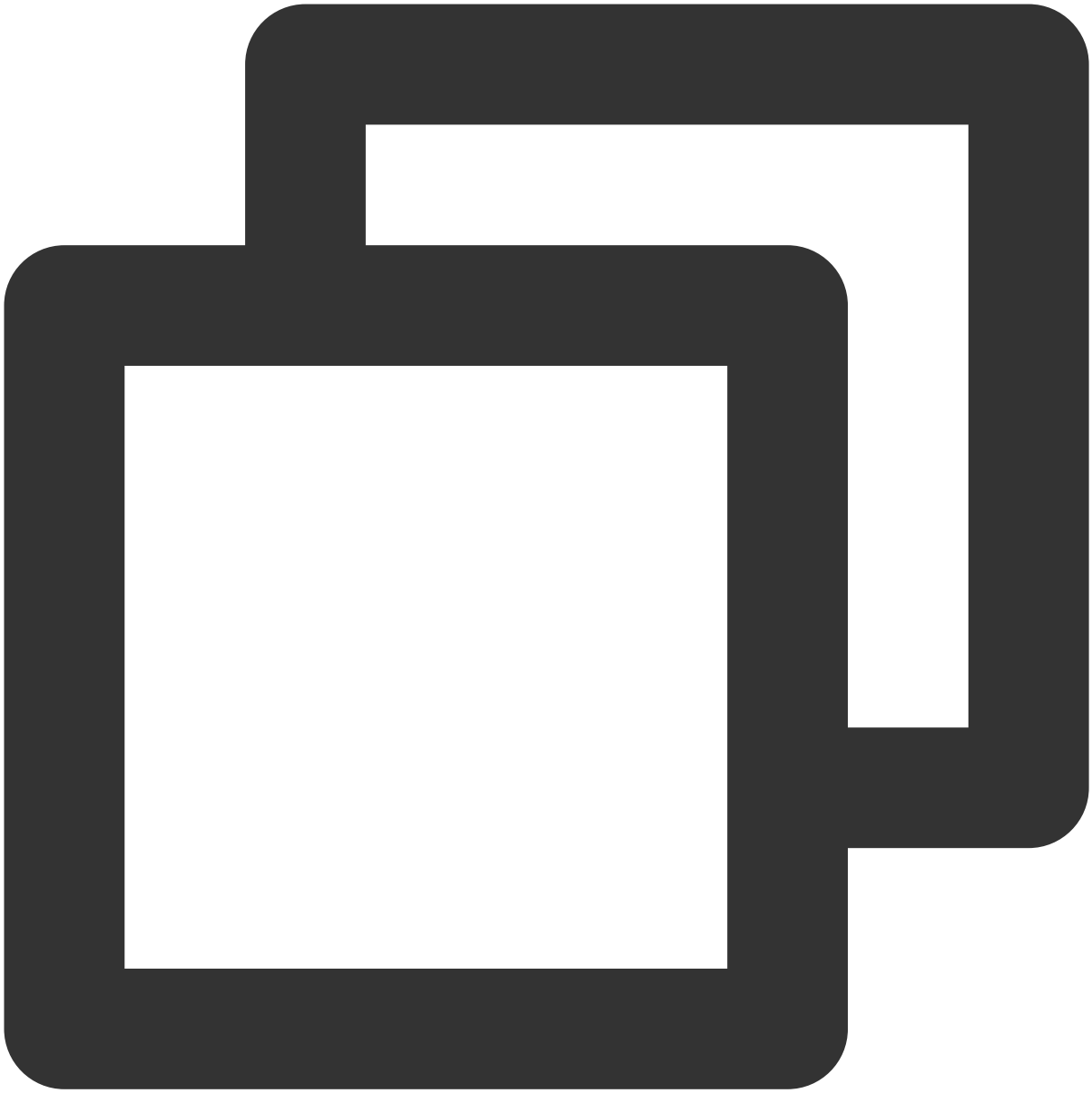
팀 기반 시나리오

팀 기반 시나리오에는 팀 생성, 임시 팀 탈퇴, 팀 리더 되기, 다른 사람을 팀에 가입하도록 초대 및 팀 해산이 포함됩니다. 다음은 다양한 시나리오의 구현을 설명하는 몇 가지 코드 예시입니다.
게임 시작 전 팀 생성: 첫 번째 플레이어가 게임에 입장하면 서버에서 자동으로 그룹을 생성하고 최대 그룹 구성원 수를 지정할 수 있습니다. 요청이 그룹 소유자 또는 그룹 구성원을 지정하는 경우 지정된 그룹 소유자 또는 그룹 구성원은 그룹이 생성될 때 그룹에 자동으로 추가됩니다. 다음은 예시 요청 URL입니다.



```
https://console.tim.qq.com/v4/group_open_http_svc/create_group?sdkappid=88888888&id
```

다음은 기본 요청 예시입니다.

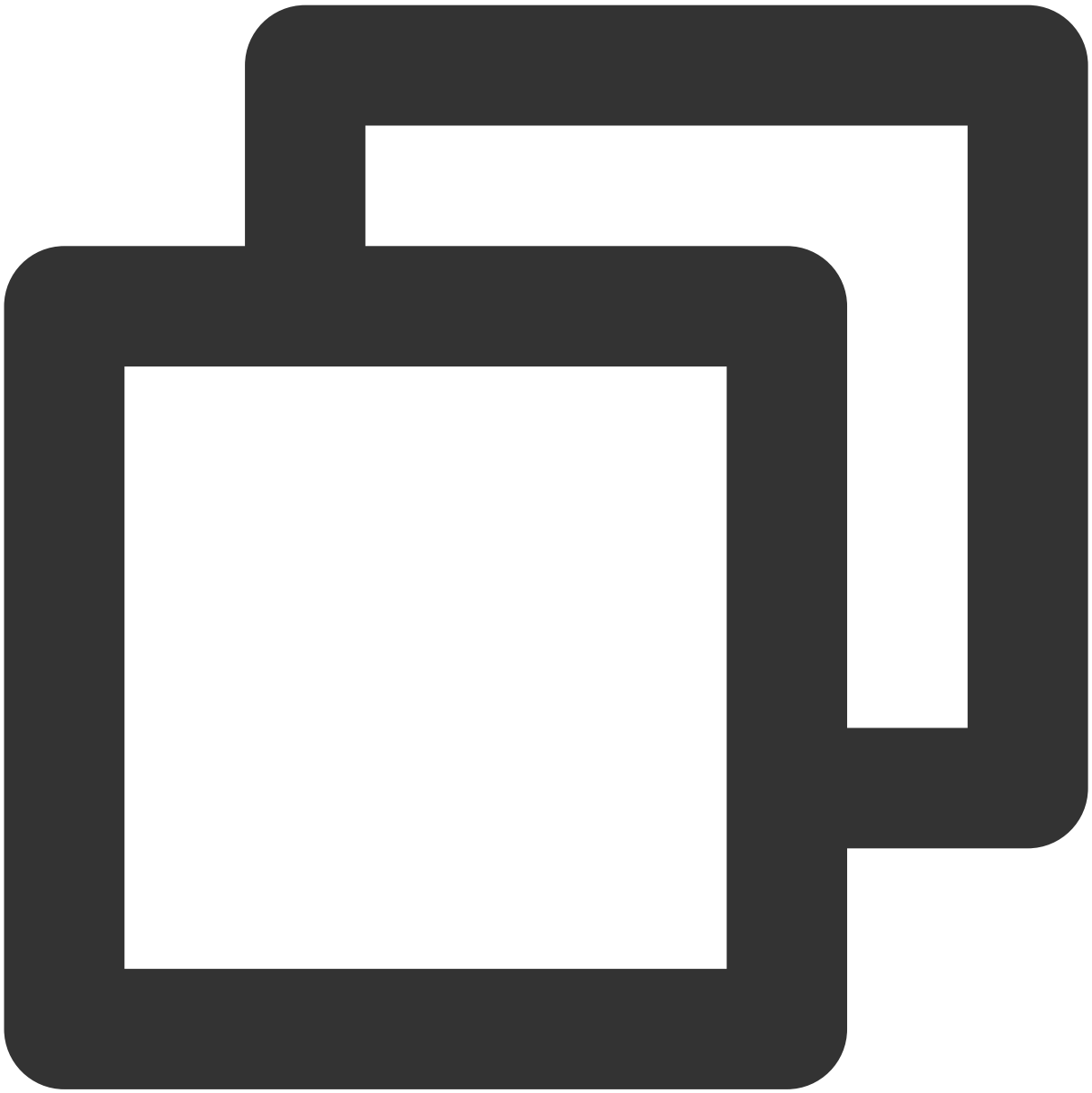


```
{
  "Owner_Account": "leckie", // 그룹 소유자의 UserId(선택 사항)
  "Type": "Public", // 그룹 유형: Private/Public/ChatRoom/AVChatRoom/Community
  "Name": "TestGroup", // 그룹 이름(필수)
  "MaxMemberCount":5 // 최대 그룹 구성원 수(선택 사항)
}
```

설명 :

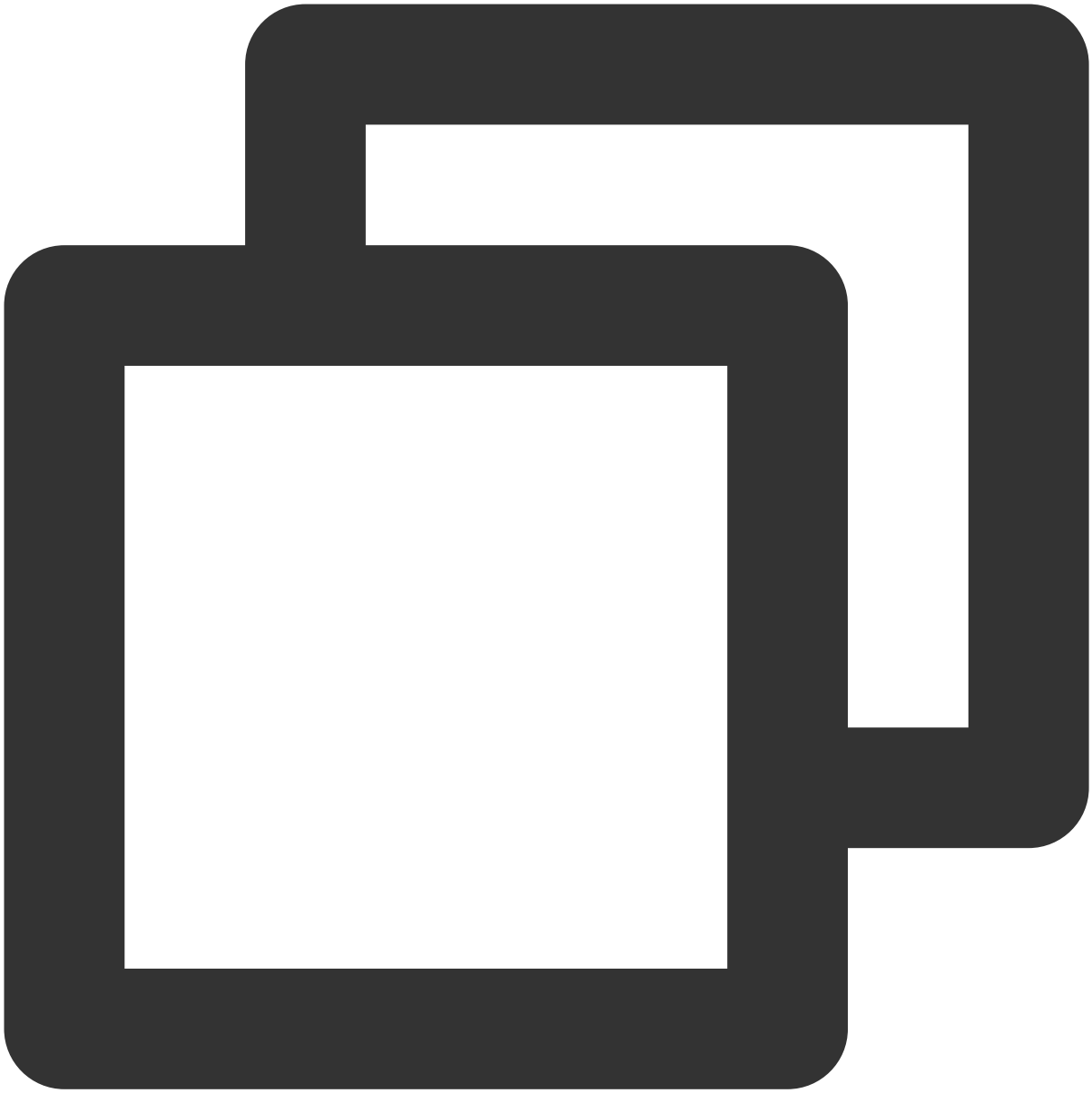
App은 최대 10만개의 그룹을 지원합니다. 추가 그룹의 경우 요금 안내에 설명된 대로 특정 요금을 지불해야 합니다. 서버 측 그룹 생성에 대한 자세한 내용은 Creating a Group을 참고하십시오.

그룹 구성원 추가: 그룹 채팅이 생성된 후 새로운 플레이어가 게임에 입장하는 경우 기존 그룹에 새로운 플레이어를 추가해야 합니다. 다음은 예시 요청 URL입니다.



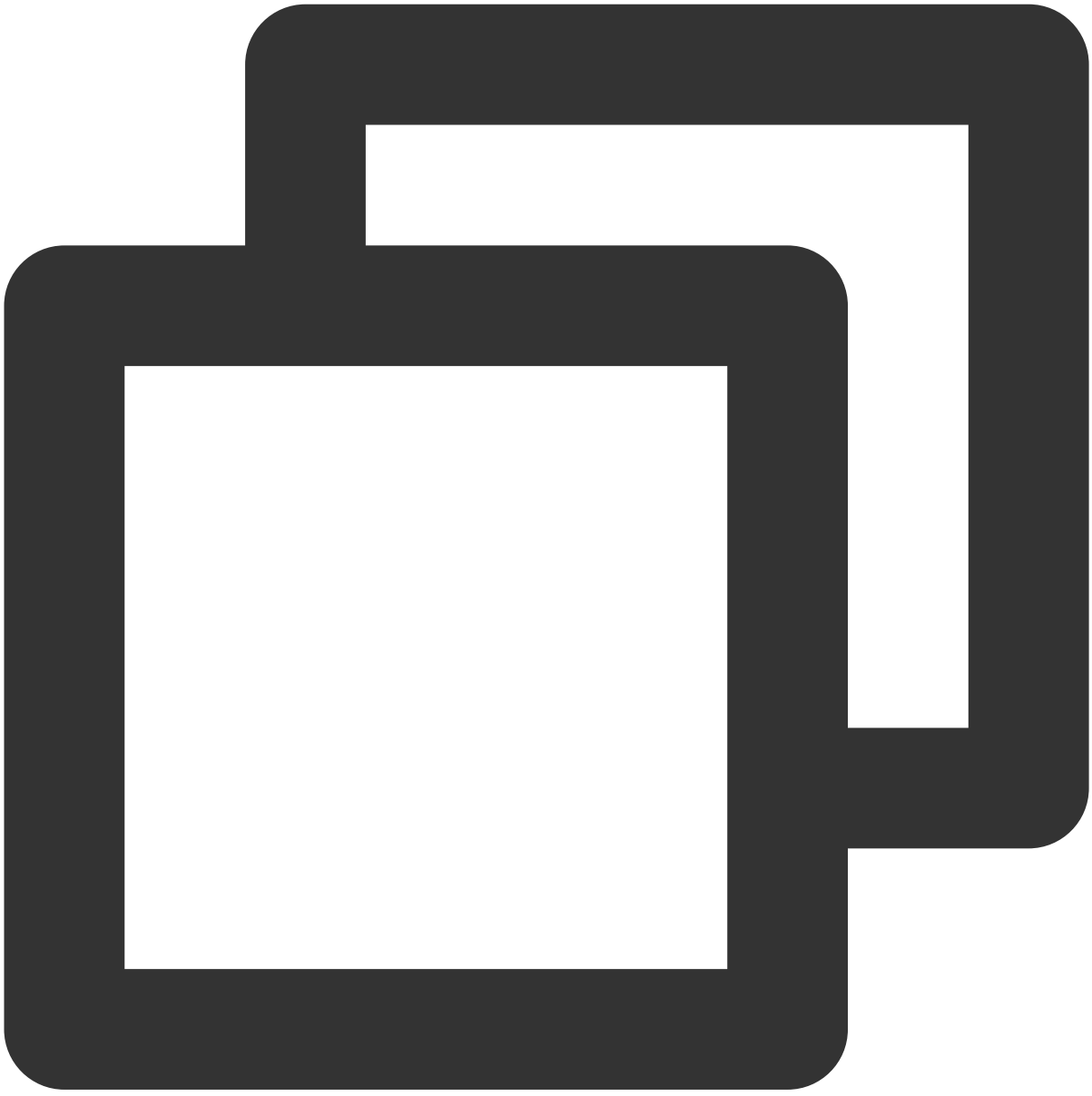
```
https://console.tim.qq.com/v4/group_open_http_svc/add_group_member?sdkappid=8888888
```

다음은 요청 샘플입니다.



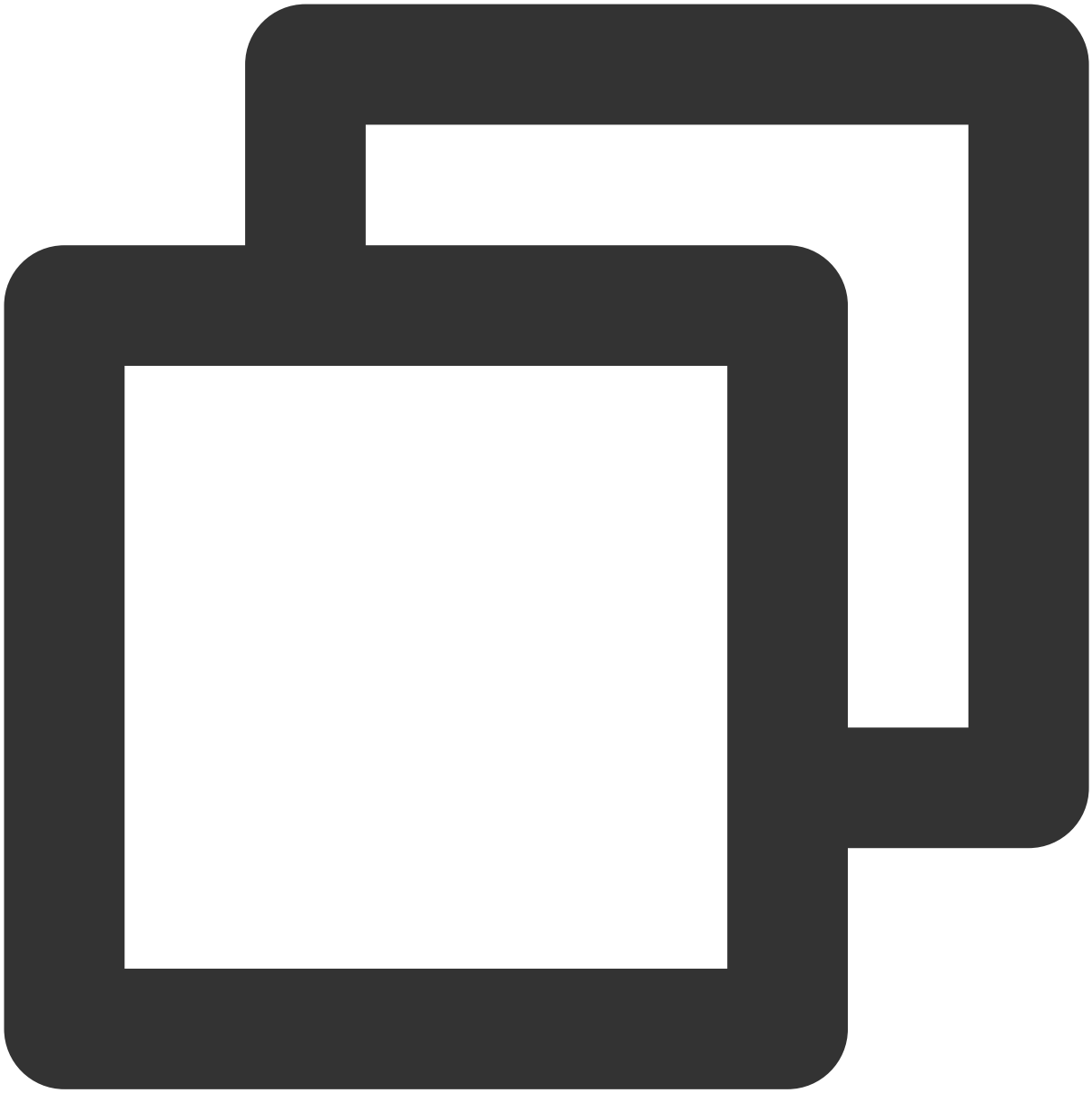
```
{
  "GroupId": "@TGS#2J4SZEAEL", // 구성원을 추가할 그룹(필수)
  "MemberList": [ // 한 번에 최대 300명의 구성원 추가 가능
      {
          "Member_Account": "tommy" // 추가할 구성원의 ID(필수)
      },
      {
          "Member_Account": "jared"
  }]
}
```

자세한 내용은 Adding Group Members를 참고하십시오.
성공적인 팀업을 위한 콜백: 그룹 생성 시 최대 그룹 구성원 수를 설정하면 그룹에 필요한 구성원 수가 모두 있어야 게임을 시작할 수 있습니다. `그룹이 가득 찬 후 콜백` 을 수신하면 게임을 시작할 수 있습니다. 다음은 예시 요청 URL입니다.



```
https://www.example.com?SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&content
```

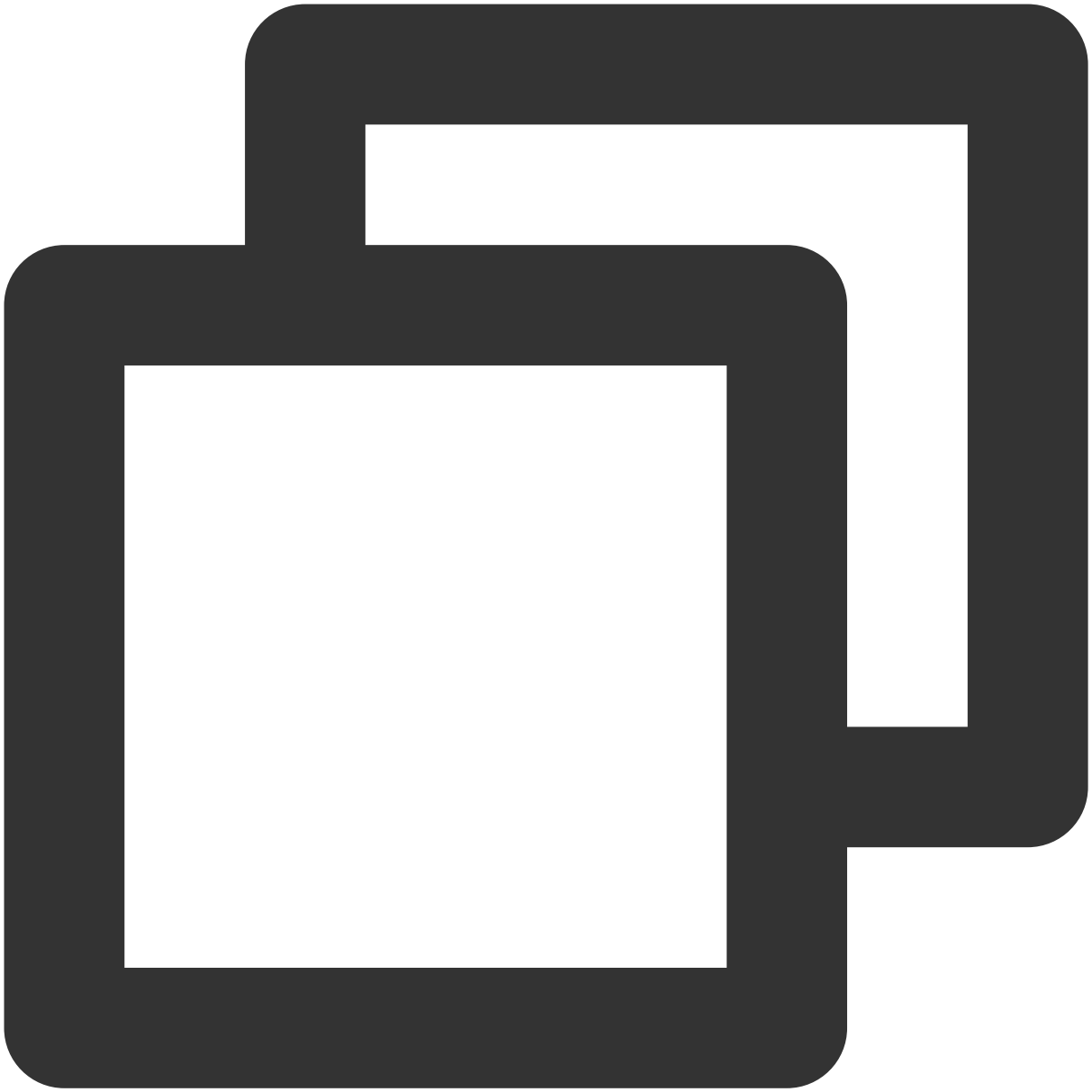
다음은 예시 요청입니다.



```
{
  "CallbackCommand": "Group.CallbackAfterGroupFull", // 콜백 명령
  "GroupId": "@TGS#2J4SZEAEL" // 그룹 ID
}
```

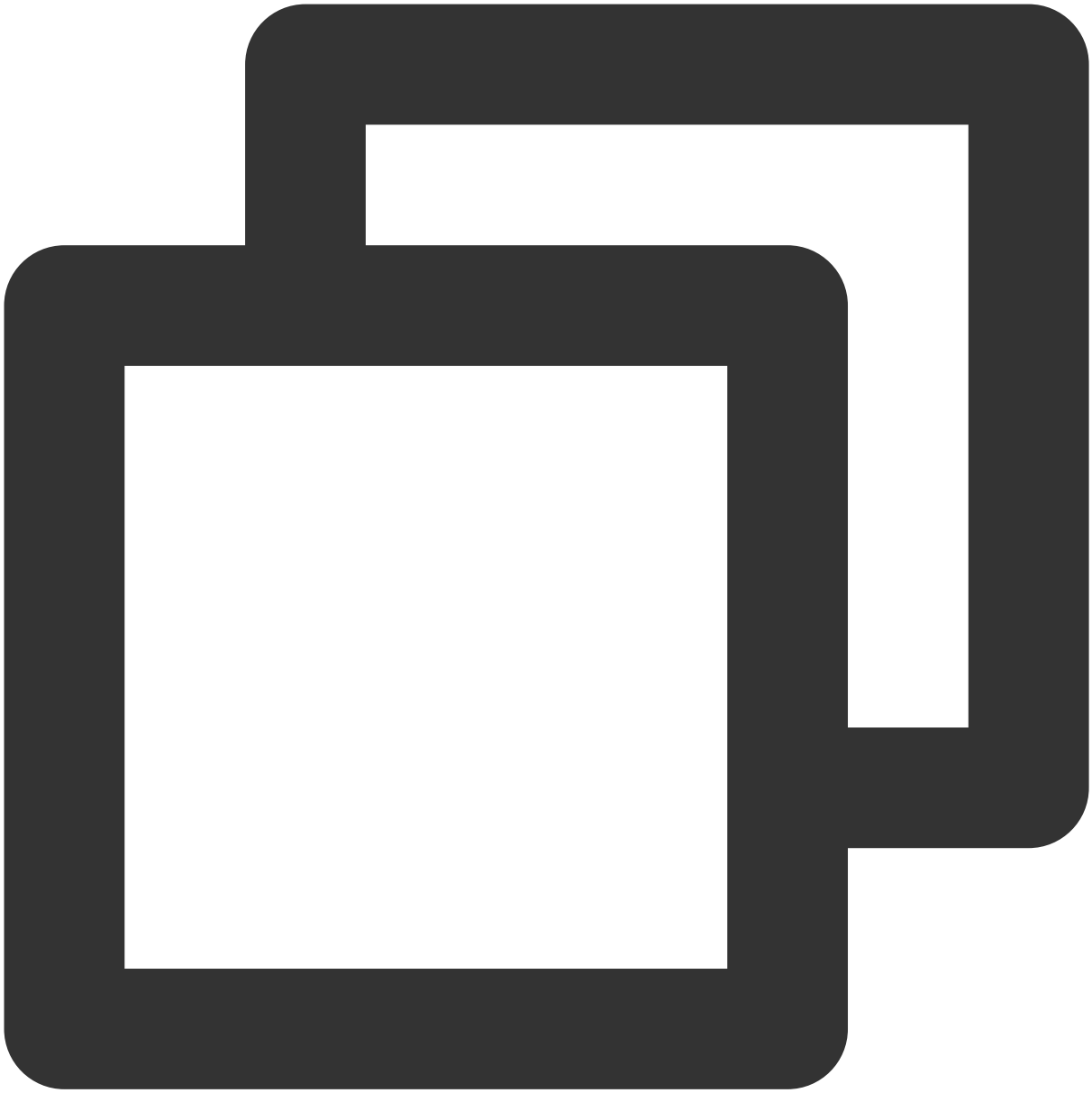
자세한 내용은 그룹 만원 후 콜백을 참고하십시오.

그룹 가입 신규 구성원 알림: 신규 플레이어가 게임(그룹 채팅)에 입장하면 `사용자가 그룹에 가입한 후 콜백` 를 시스템에서 전송하여 다른 그룹 회원에게 알립니다. 다음은 예시 요청 URL입니다.



```
https://www.example.com?SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&content
```

다음은 예시 요청입니다.



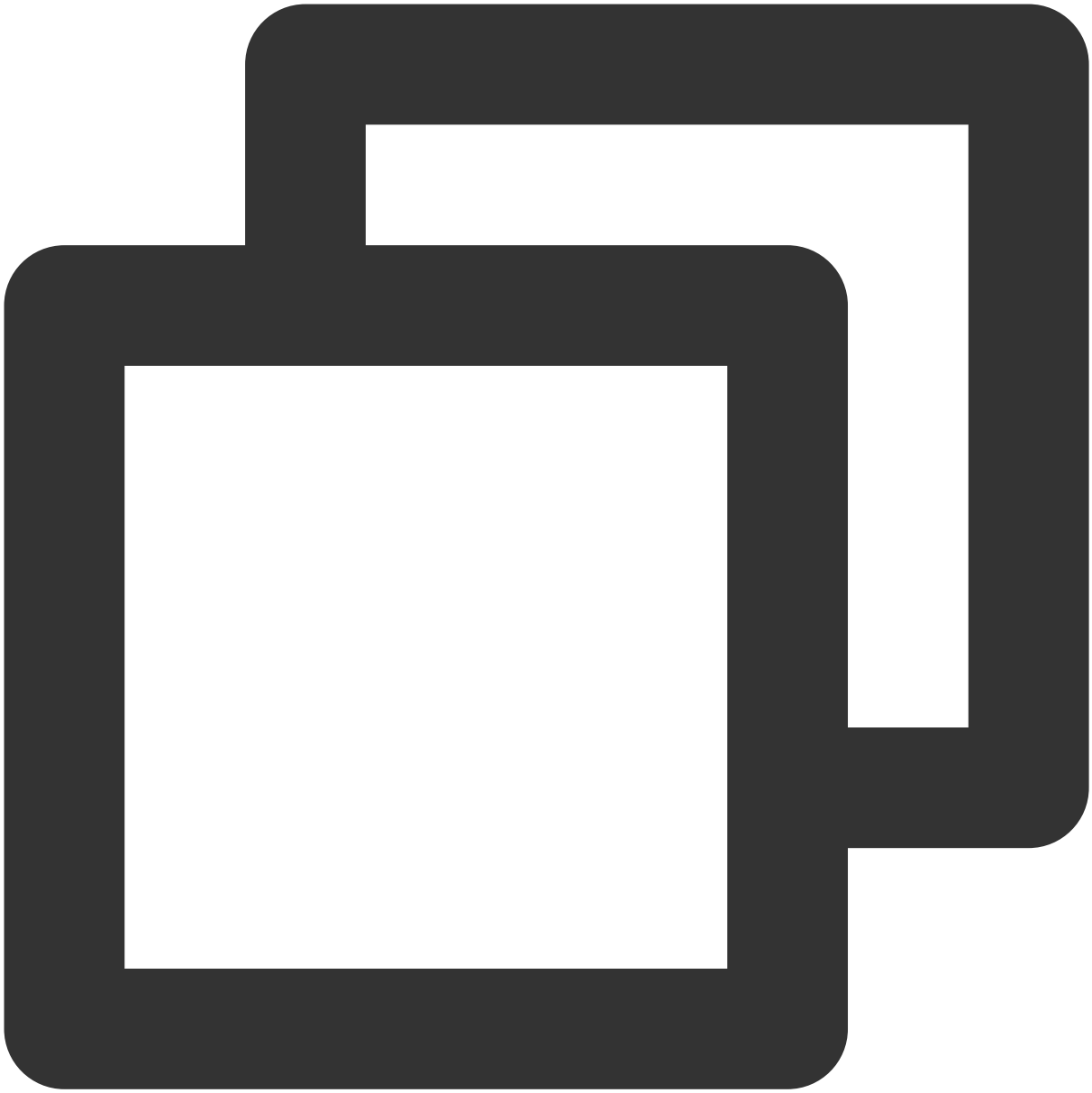
```
{
  "CallbackCommand": "Group.CallbackAfterNewMemberJoin", // 콜백 명령
  "GroupId" : "@TGS#2J4SZEAEL",
  "Type": "Public", // 그룹 유형
  "JoinType": "Apply", // 그룹 가입 방법: Apply(신청); Invited(초대)
  "Operator_Account": "leckie", // 운영자
  "NewMemberList": [ // 새 구성원 목록
      {
          "Member_Account": "jared"
      },
      {
```

```
            "Member_Account": "tommy
        }
    ]
}
```

설명：

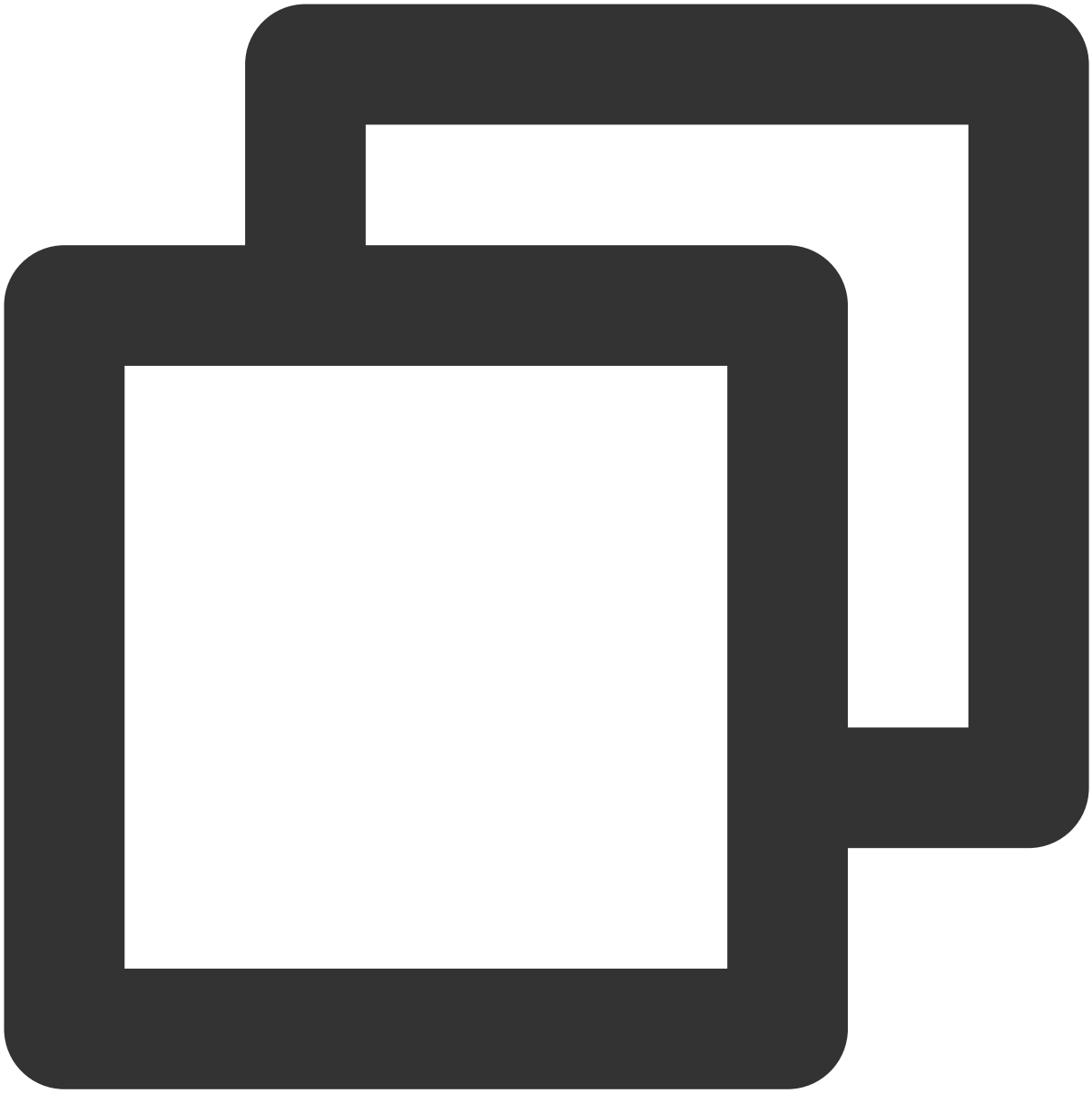
이 콜백을 활성화하려면 콜백 URL을 구성하고 해당 프로토콜을 토글해야 합니다. 구성 방법에 대한 자세한 내용은 서드파티 콜백 설정을 참고하십시오. 자세한 내용은 After a User Joins a Group을 참고하십시오.

게임 중 팀 탈퇴: 플레이어가 자발적으로 게임을 떠나거나 네트워크 문제로 인해 게임을 떠나는 경우, 서버는 `사용자가 그룹을 탈퇴한 후 콜백` 을 통해 누군가가 그룹 채팅에서 탈퇴했음을 그룹의 다른 플레이어에게 알릴 수 있으며, 네트워크 문제로 인해 게임을 탈퇴한 플레이어에게 메시지를 보낼 수도 있습니다. 다음은 After a User Leaves a Group의 요청 URL 예시입니다.



```
https://www.example.com?SdkAppid=$SDKAppID&CallbackCommand=$CallbackCommand&content
```

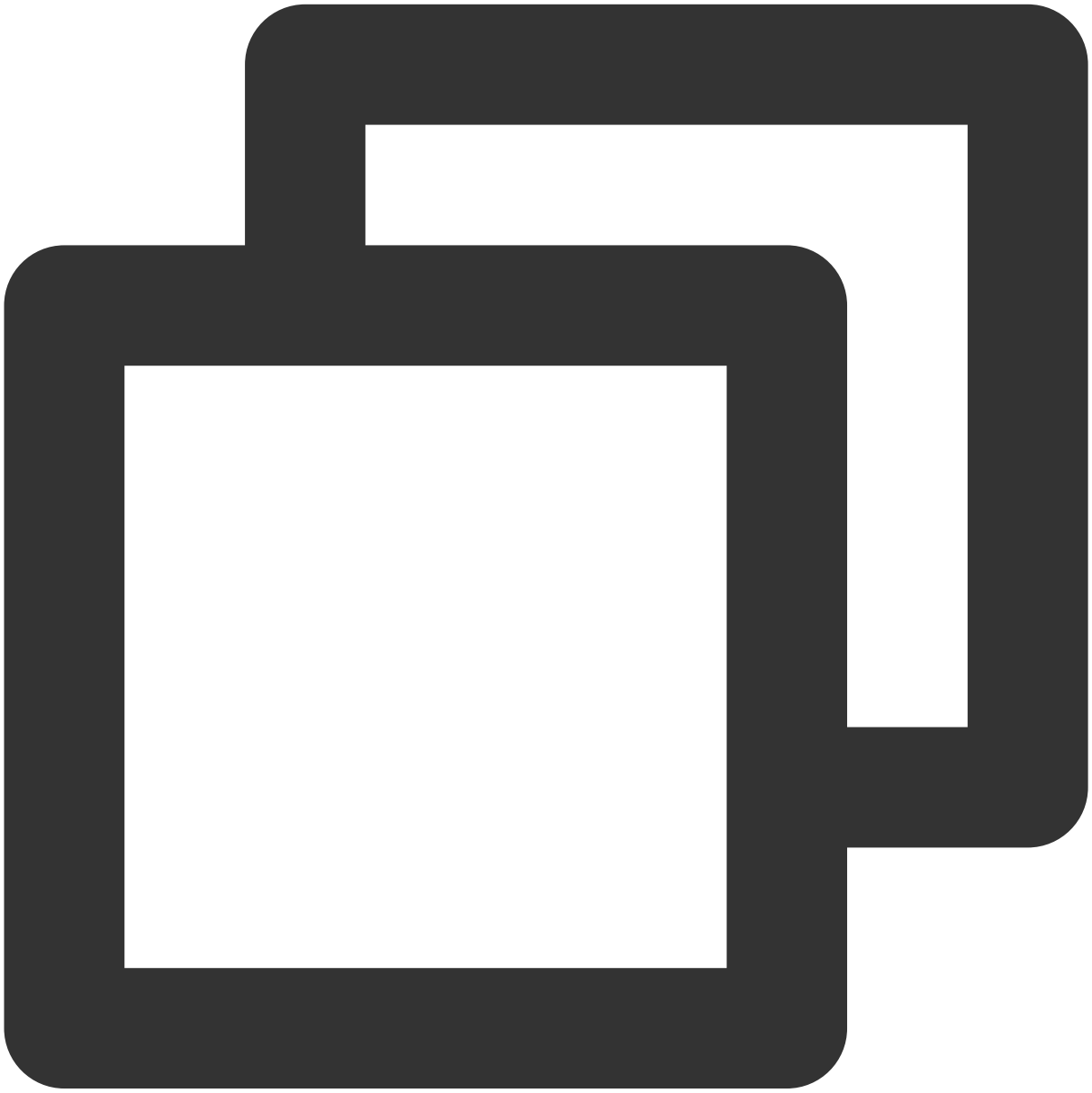
다음은 예시 요청입니다.



```
{
  "CallbackCommand": "Group.CallbackAfterMemberExit", // 콜백 명령
  "GroupId": "@TGS#2J4SZEAEL", // 그룹 ID
  "Type": "Public", // 그룹 유형
  "ExitType": "Kicked", // 구성원 이탈 유형: Kicked-퇴장 당함; Quit-자발적 퇴장
  "Operator_Account": "leckie", // 운영자
  "ExitMemberList": [ // 그룹을 탈퇴한 구성원 목록
      {
          "Member_Account": "jared"
      },
      {
```

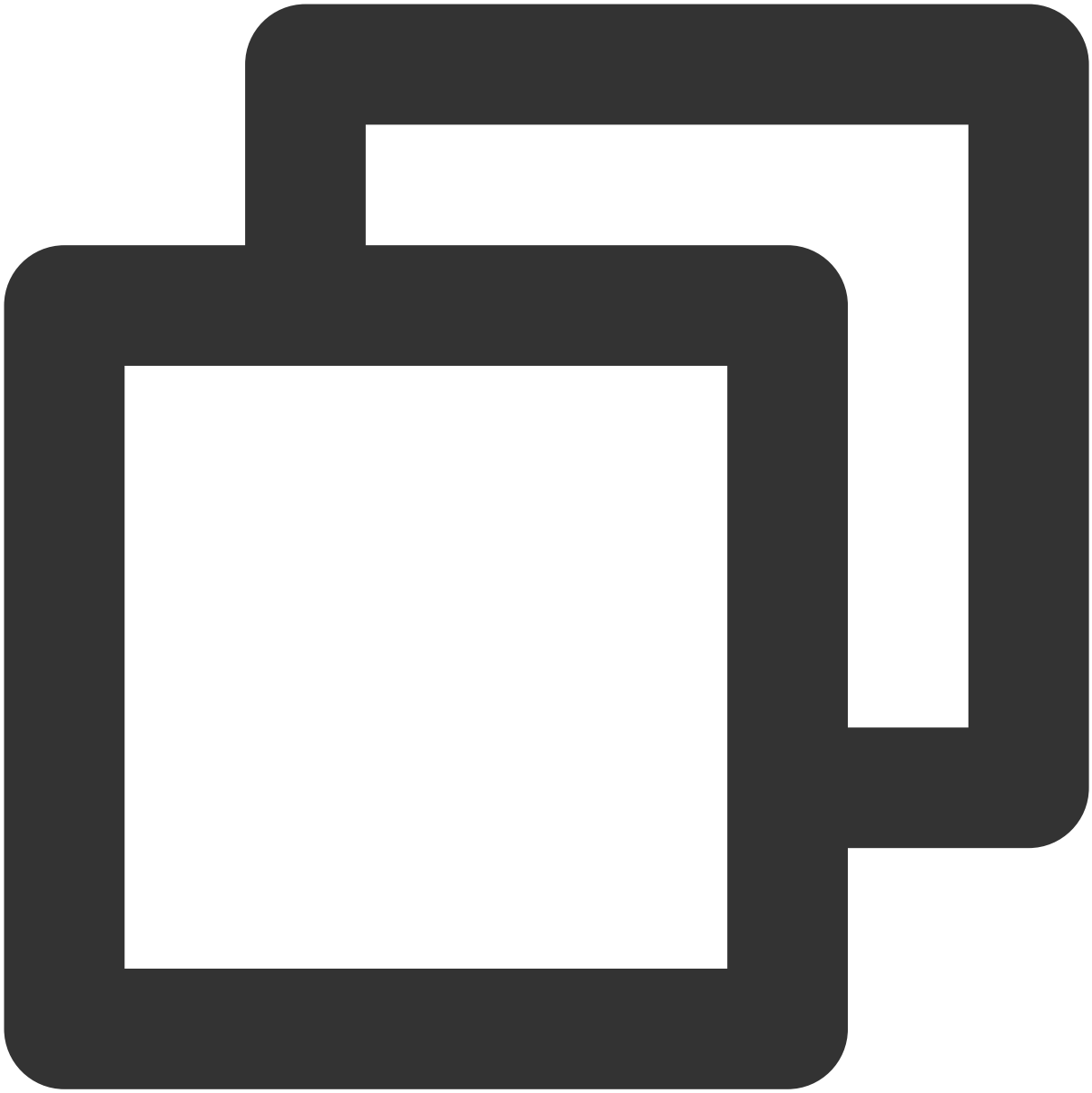
```
            "Member_Account": "tommy"
        }
    ]
}
```

게임 종료 후 단체 채팅 해산: 게임 종료 후 서버 측에서 직접 단체 대화를 해산할 수 있습니다. 다음은 Disbanding a Group에 대한 요청 URL 예시입니다.



```
https://console.tim.qq.com/v4/group_open_http_svc/destroy_group?sdkappid=88888888&i
```

다음은 예시 요청입니다.

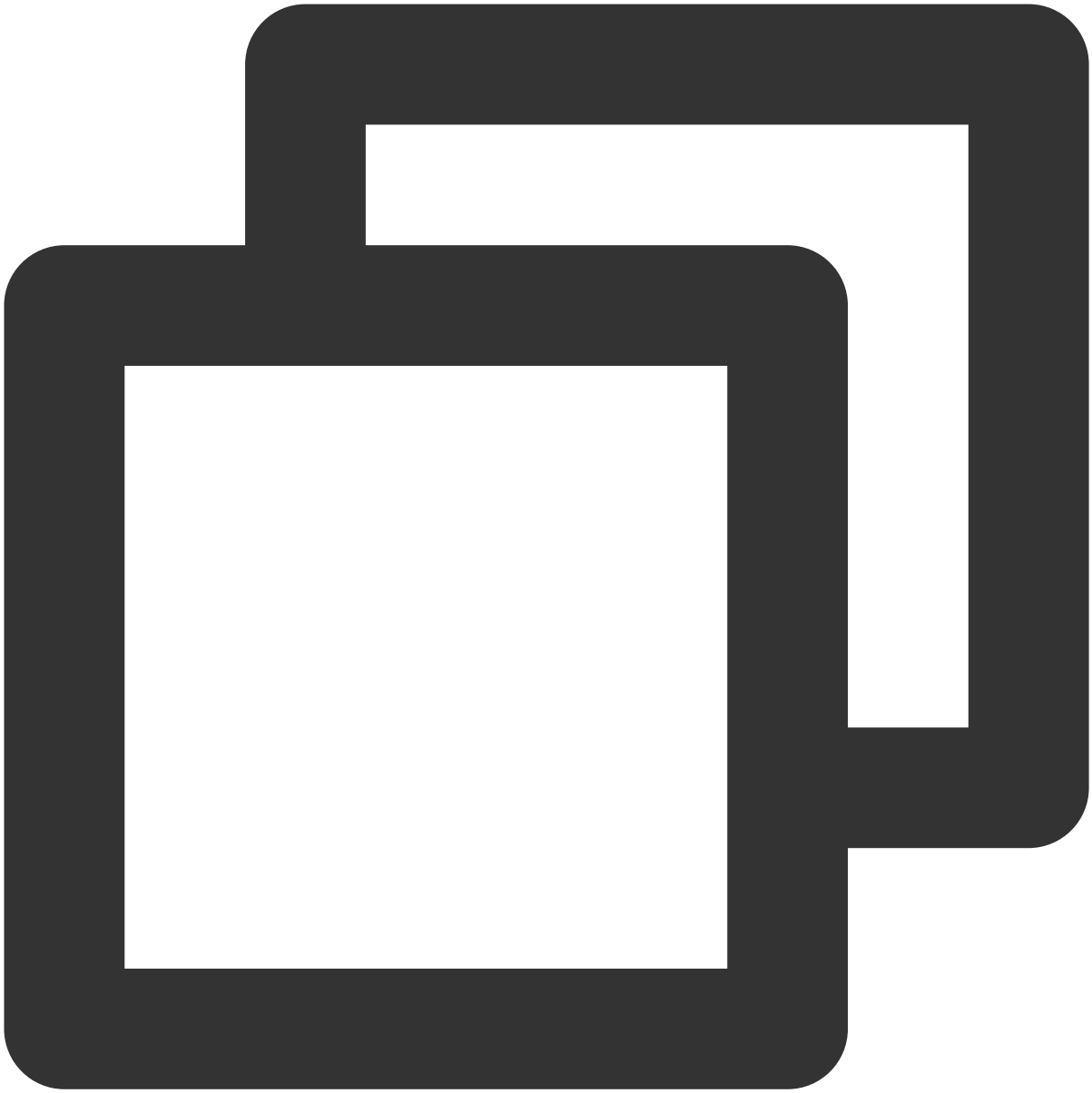


```
{
  "GroupId": "@TGS#2J4SZEAEL"
}
```

임시 팀의 오디오/비디오 채팅은 복잡하며 아래에서 자세히 설명합니다.

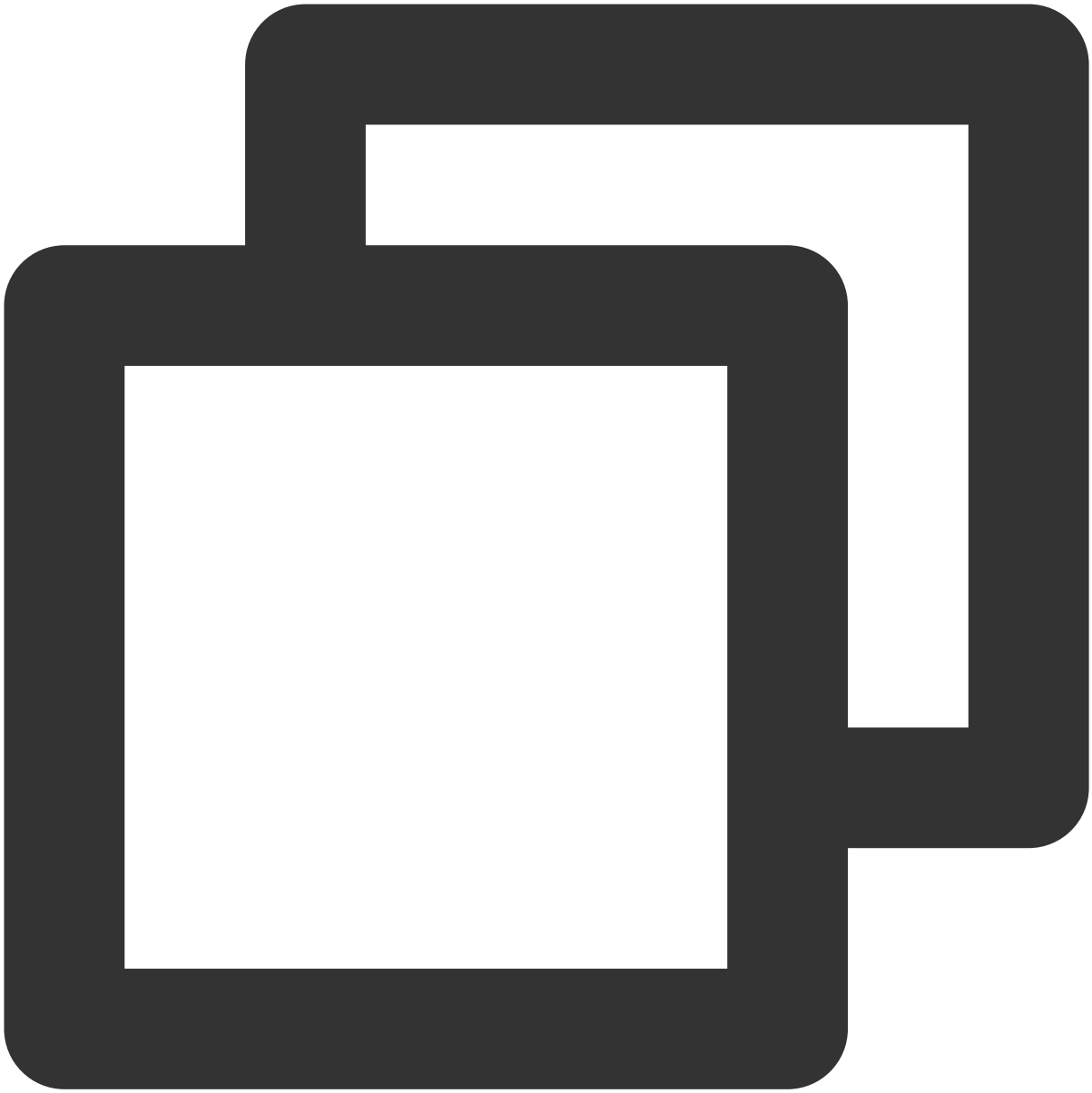
백그라운드에서 팀 정보 가져오기

그룹 세부 정보 가져오기: 서버 API `그룹 프로필 가져오기` 를 호출하여 그룹의 구성원 수 및 그룹 구성원의 기본 정보와 같은 그룹 세부 정보를 가져올 수 있습니다. 요청에서 Filter 필드를 사용하여 가져올 정보를 필터링할 수 있습니다. 다음은 샘플 요청 URL입니다.



```
https://console.tim.qq.com/v4/group_open_http_svc/get_group_info?sdkappid=88888888&
```

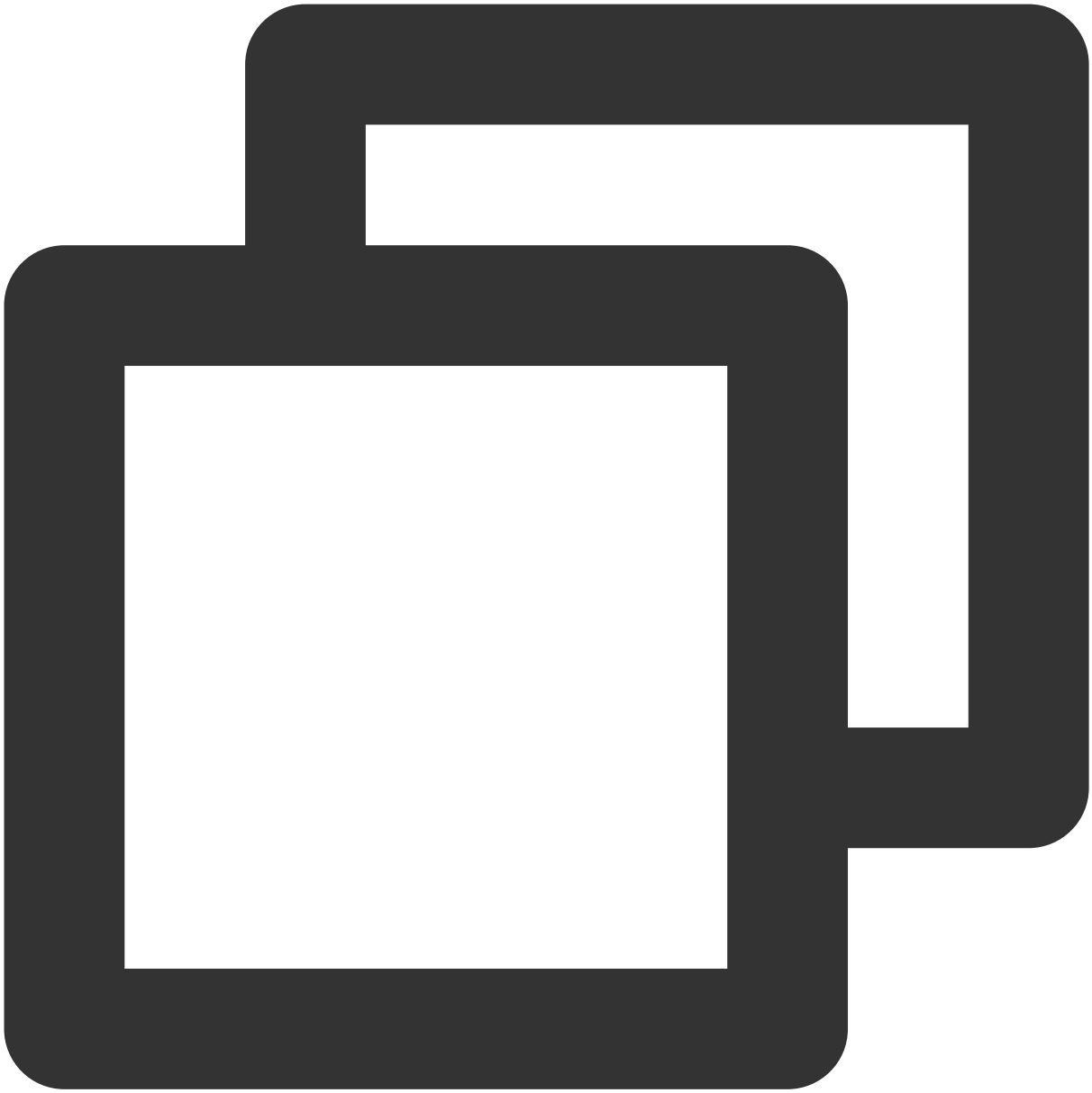
다음은 예시 요청입니다.



```
{
  "GroupIdList": [ // 쿼리에 지정된 그룹 ID 목록(필수)
      "@TGS#1NVTZEAE4",
      "@TGS#1CXTZEAET"
  ]
}
```

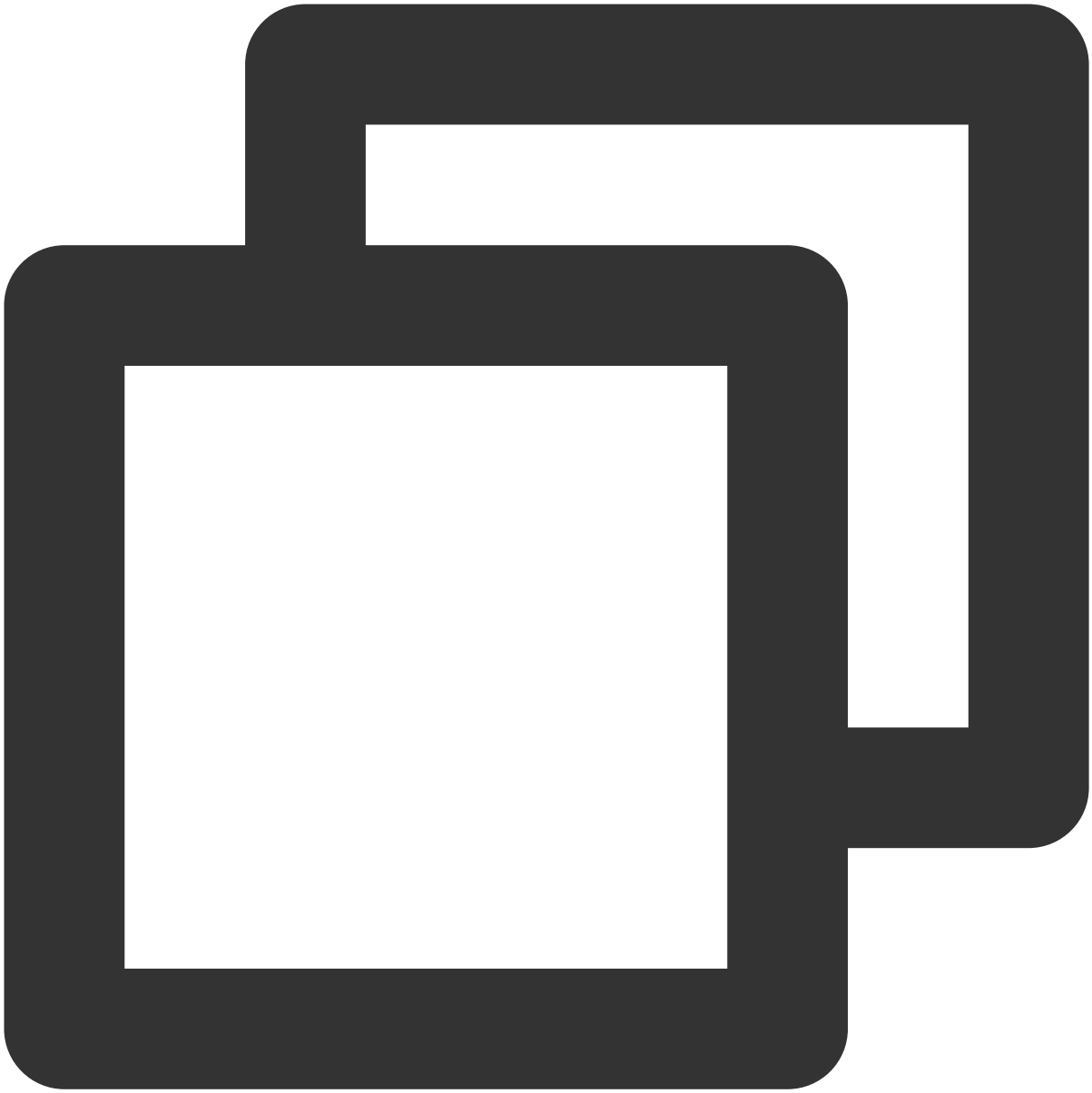
자세한 내용은 Getting Group Profiles를 참고하십시오.

그룹 구성원 세부 정보 가져오기: `그룹 구성원 프로필 가져오기` API를 호출하여 그룹 구성원의 세부 정보(사용자 지정 필드 포함)를 가져올 수 있습니다. 다음은 샘플 요청 URL입니다.



```
https://console.tim.qq.com/v4/group_open_http_svc/get_group_member_info?sdkappid=88
```

다음은 예시 요청입니다.



```
{
  "GroupId":"@TGS#1NVTZEAE4"   // 그룹 ID(필수)
}
```

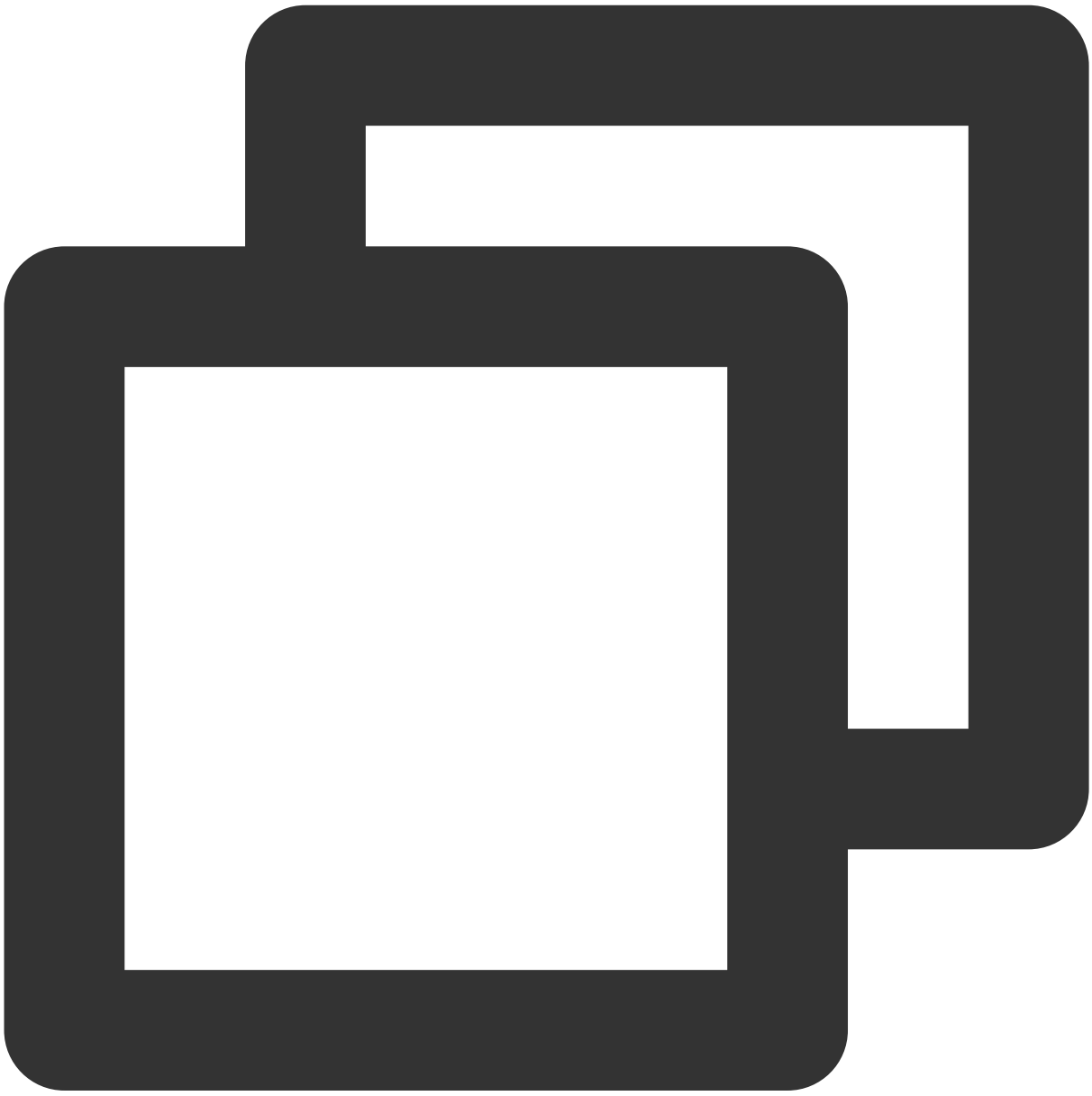
응답에서 `MemberNum` 필드는 그룹의 총 구성원 수를 나타내고 `AppMemberDefinedData` 는 그룹 구성원의 사용자 지정 필드 정보를 나타냅니다.

설명：

이 API는 오디오-비디오 그룹(AVChatRoom)을 지원하지 않습니다. 자세한 내용은 Getting Group Member Profiles를 참고하십시오.

팀 구성원의 팀 정보 가져오기

팀 구성원은 `그룹 구성원 프로필 가져오기` API를 호출하여 팀 내 다른 구성원의 역할 및 상태와 같은 자세한 정보를 얻을 수 있습니다.



```
GroupGetMemberInfoListParam param = new GroupGetMemberInfoListParam
{
  group_get_members_info_list_param_group_id = "group_id",
  group_get_members_info_list_param_identifier_array = new List<string>
  {
    "user_id"
  }
```

```
};
TIMResult res = TencentIMSDK.GroupGetMemberInfoList(param, (int code, string desc,
 // 비동기 로직 프로세스
});
```

설명：

자세한 내용은 Unity-Getting the Profile of a Group Member를 참고하십시오.

`그룹 만원 시 게임 시작`, `구성원의 팀 가입 또는 탈퇴` 등과 같은 기타 그룹 정보는 팀 기반 시나리오를 참고하십시오. 이러한 정보는 콜백을 통해 그룹에 통보됩니다.

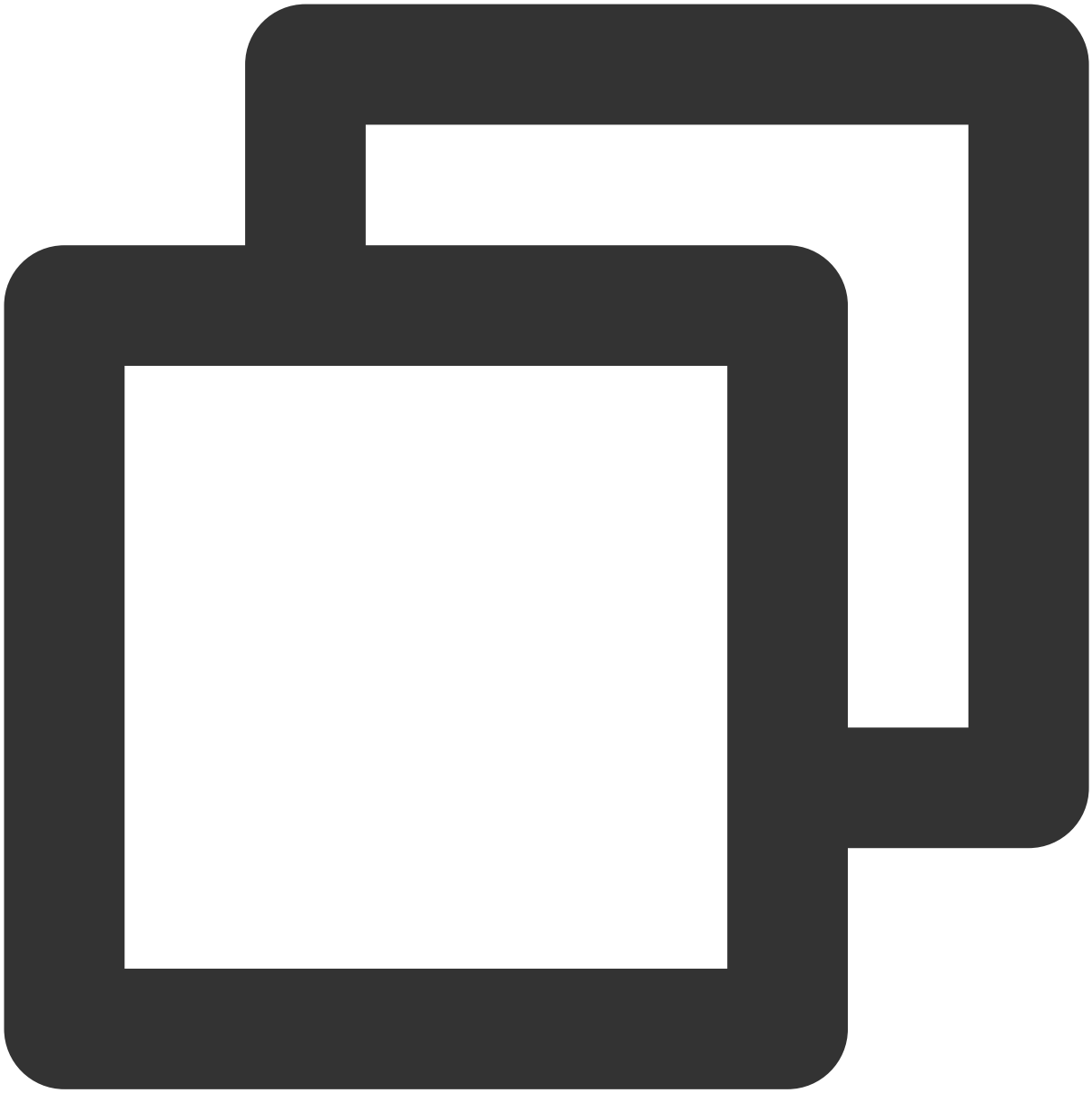
그룹 선택 및 기타 그룹 관련 정보에 대한 자세한 내용은 그룹 시스템을 참고하십시오. **그룹 관리**에 대한 자세한 내용은 콘솔 가이드 - 그룹 관리를 참고하십시오.

민감한 정보 필터링

민감한 콘텐츠를 필터링하는 것은 게임의 중요한 기능입니다. 구현 계획은 다음과 같습니다.

1. 그룹 메시지를 보내기 전에 웹후크를 바인딩합니다.
2. 웹후크 데이터를 기반으로 메시지 유형을 식별하고 메시지 데이터를 Tencent Cloud Security 또는 다른 타사 점검 서비스에 전달합니다.
3. 메시지가 일반 텍스트 메시지인 경우 Tencent Cloud Security의 점검 결과를 기다린 후 메시지를 IM 백엔드로 전달할지 여부를 나타내는 데이터 패킷을 반환합니다.

메시지를 보내기 전 웹후크의 샘플 데이터:



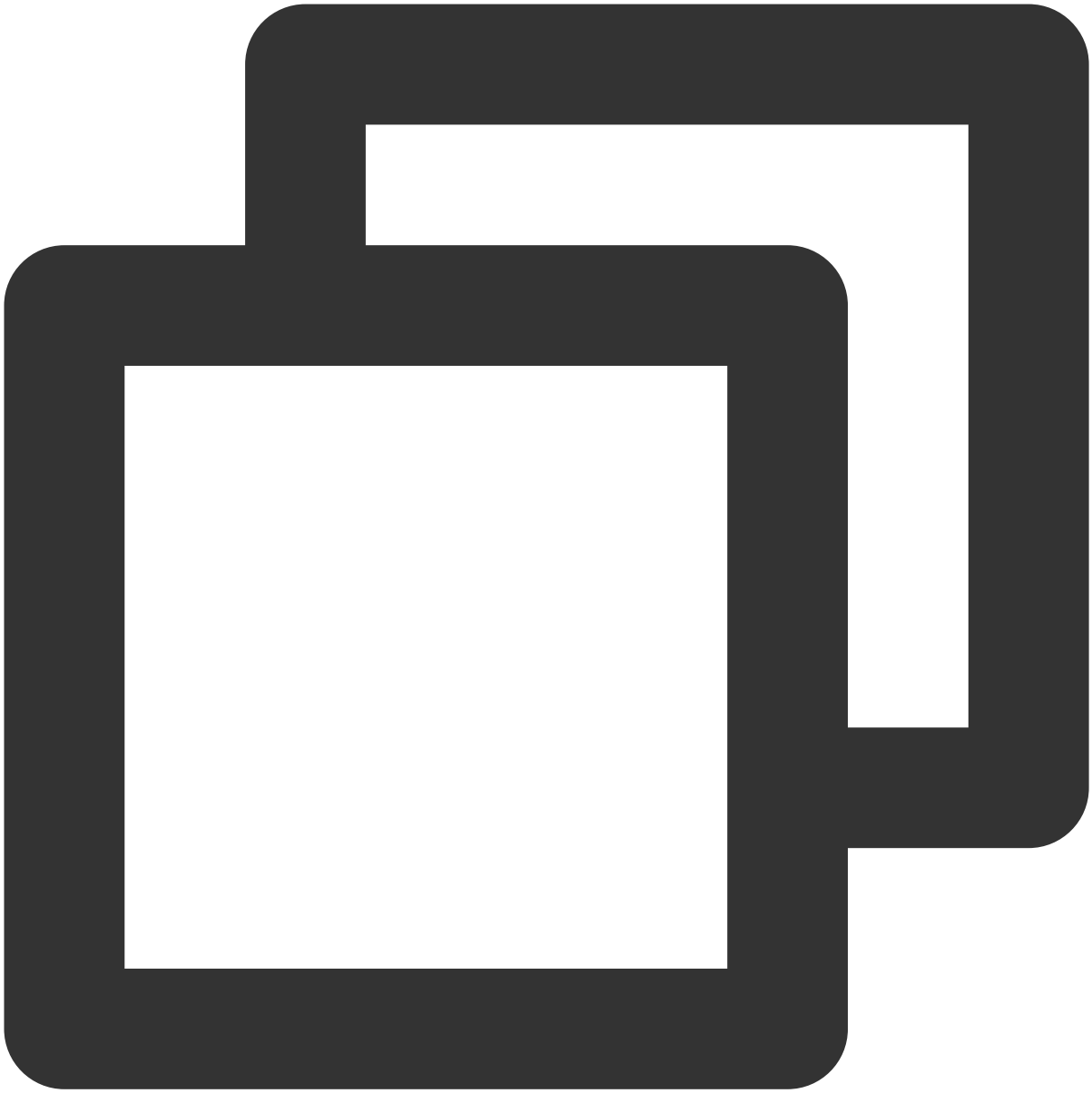
```
{
    "CallbackCommand": "Group.CallbackBeforeSendMsg", // 콜백 명령
    "GroupId": "@TGS#2J4SZEAEL", // 그룹 ID
    "Type": "Public", // 그룹 유형
    "From_Account": "jared", // 발신자
    "Operator_Account":"admin", // 요청 개시자
    "Random": 123456, // 랜덤 숫자
    "OnlineOnlyFlag": 1, // 값은 온라인 메시지인 경우 1이고 그렇지 않은 경우 0(기본값)입니다
    "MsgBody": [ // 메시지 본문, 자세한 내용은 TIMMessage 메시지 객체 참고
        {
            "MsgType": "TIMTextElem", // 텍스트
```

```
            "MsgContent": {
                "Text": "red packet"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data"
}
```

설명：

MsgBody의 MsgType 필드를 기반으로 메시지 유형을 식별할 수 있습니다. 필드에 대한 자세한 내용은 그룹 내 발언 전 콜백을 참고하십시오.

IM 백엔드로 반환된 웹후크의 데이터 패킷을 통해 구현될 수 있는 특정 방식으로 비준수 메시지를 처리하도록 선택할 수 있습니다.



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0 // 다른 ErrorCode 값은 다른 의미를 갖습니다
}
```

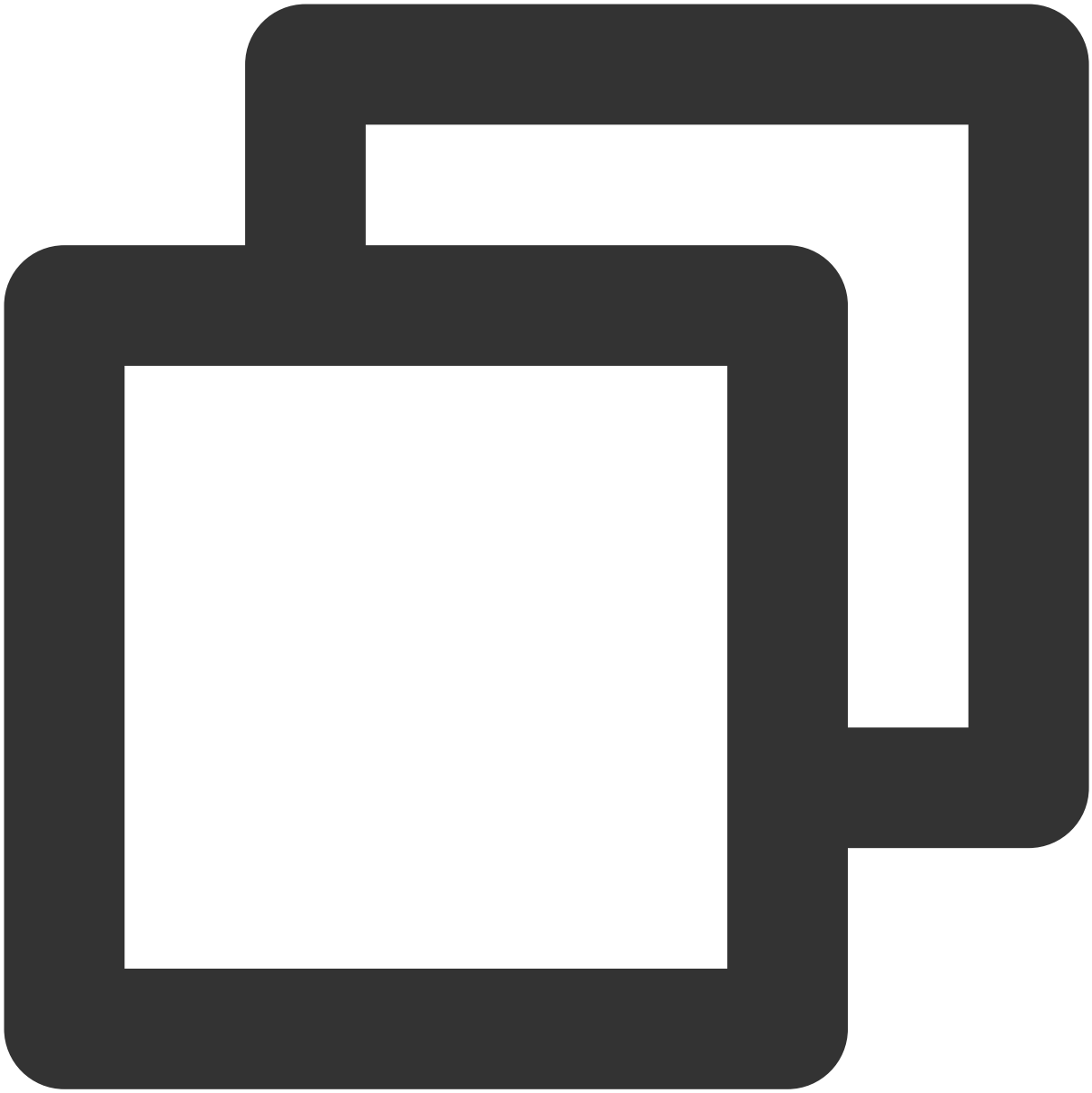
ErrorCode	설명
0	발언 허용, 메시지 전달 가능
1	발언 거부, 클라이언트 10016 반환

2	자동 폐기가 활성화되고 클라이언트가 메시지를 정상적으로 반환

설명：
필요에 따라 사용할 수 있습니다.

메시지 유형 사용자 지정(소품 선물 및 거래 메시지 등)

게임 채팅에는 텍스트, 이모티콘, 음성 메시지와 같은 단순 메시지 외에 사용자 지정 메시지가 필요합니다. 사용자 정의 메시지를 통해 개발자는 게임 소품 선물 및 거래와 같은 더 많은 대화방 기능을 구현하기 위해 메시지의 콘텐츠 형식을 사용자 정의할 수 있습니다.
Tencent Cloud IM은 텍스트, 이모티콘, 지리적 위치, 이미지, 음성, 파일, UGSV, 시스템 알림 및 사용자 지정 메시지의 9가지 기본 메시지 유형을 제공합니다. 사용자 지정 메시지를 제외한 모든 메시지의 형식은 고정되어 있으며 사용자는 해당 정보만 입력하면 됩니다. 메시지 유형에 대한 자세한 설명은 메시지 유형을 참고하십시오. 메시지 형식에 대한 자세한 내용은 메시지 형식을 참고하십시오.
서버 API Sending One-to-One Messages to One User, Sending One-to-One Messages to Multiple Users 및 Sending Ordinary Messages in a Group을 통해 사용자에게 사용자 지정 메시지를 보낼 수 있습니다. 다음은 그룹에서 일반 메시지를 보내기 위한 기본 요청 샘플입니다.



```
{
    "GroupId": "@TGS#2C5SZEAEF",
    "Random": 8912345, // 임의의 숫자. 두 메시지의 임의의 숫자가 5분 이내에 동일하면 동일한
    "MsgBody": [ // element 배열로 구성된 메시지 본문. 자세한 내용은 필드 설명 참고.
        {
            "MsgType": "TIMTextElem", // 텍스트
            "MsgContent": {
                "Text": "red packet"
            }
        },
        {
```

```
            "MsgType": "TIMFaceElem", // 이모티콘
            "MsgContent": {
            "Index": 6,
            "Data": "abc\\u0000\\u0001"
            }
        }
    ],
    "CloudCustomData": "your cloud custom data",
    "SupportMessageExtension": 0,
}
```

`CloudCustomData` 를 수정하여 사용자 지정 메시지 데이터(클라우드에 저장됨)를 정의하고 보낼 사용자 지정 메시지를 `MsgBody` (메시지 본문)에 입력할 수 있습니다.

팀 내 오디오/비디오 채팅

게임에서 오디오/비디오 채팅은 중요한 기능입니다. Tencent Cloud IM 앱은 Tencent Real-Time Communication(TRTC) 및 TUICallKit을 사용하여 실시간 음성/영상 통화 기능을 갖추고 있습니다.

설명：

각 SDKAppID에 대해 오디오/비디오 통화 기능의 7일 무료 평가판을 제공합니다. IM 콘솔에서 각 SDKAppID에 대해 하나의 무료 평가판만 얻을 수 있습니다.

TUICallKit에 대한 자세한 내용은 Integration Solution (UI Included) - 음성/영상 통화를 참고하십시오.

게임 채팅방 유형

게임 채팅방 유형은 다음과 같습니다.

유형	특성
채널	채널은 일반적으로 많은 수의 채팅 사용자를 참여시키며 고정 구성원 목록이 없습니다. 사용자는 언제든지 채널에 가입하고 채널에서 나갈 수 있습니다. 오프라인 메시지 푸시가 필요하지 않습니다.
라이브 게임 로비	라이브 게임 로비에는 일반적으로 많은 채팅 사용자가 참여합니다. 사용자는 언제든지 라이브 룸에 참여하고 나갈 수 있습니다. 기록 메시지 쿼리가 지원됩니다.
팀	팀은 일반적으로 서로 친구가 될 필요가 없는 소수의 채팅 사용자와 관계를 맺습니다. 게임이 끝나면 팀이 종료됩니다. 오프라인 메시지 푸시가 필요하지 않습니다.
친구	C2C 채팅. 채팅 기록이 저장됩니다. 채팅 대상은 연락처 목록에 있는 친구만 될 수 있습니다.
비공개 메시지	C2C 채팅. 채팅 대상은 낯선 사람일 수 있습니다.
오디오-비디오 그룹	채팅 사용자 수에는 제한이 없습니다. 사용자는 언제든지 그룹에 가입하거나 탈퇴할 수 있습니다.

IM은 다음 유형의 채팅방을 제공합니다.

유형	특성
작업 그룹 (work)	작업 그룹은 그룹 구성원인 친구의 초대를 통해 사용자가 그룹에 가입할 수 있도록 합니다. 초대는 초대받은 사람이 수락하거나 그룹 소유자가 승인할 필요가 없습니다.
공개 그룹 (Public)	공개 그룹을 사용하면 그룹 소유자가 그룹 관리자를 지정할 수 있습니다. 그룹에 가입하려면 사용자가 그룹 ID를 검색하여 요청을 보내고 그룹 소유자 또는 관리자가 요청을 승인해야 그룹에 가입할 수 있습니다.
회의 그룹 (Meeting)	회의 그룹을 사용하면 사용자가 자유롭게 참여하고 나갈 수 있으며 그룹에 참여하기 전에 전송된 기록 메시지를 볼 수 있습니다. 회의 그룹은 음성/화상 회의 및 온라인 교육과 같은 TRTC(Tencent Real-Time Communication)를 통합하는 시나리오에 이상적입니다. 이 그룹 유형은 이전 버전의 ChatRoom과 동일합니다.
오디오-비디오 그룹 (AVChatRoom)	오디오-비디오 그룹은 사용자가 자유롭게 가입 및 탈퇴할 수 있고, 회원 수에 제한이 없으며, 메시지 기록을 저장하지 않습니다. 오디오-비디오 그룹은 Cloud Streaming Services(CSS)와 함께 사용하여 화면 댓글 채팅 시나리오를 지원할 수 있습니다.
커뮤니티 그룹 (Community)	커뮤니티 그룹은 사용자가 자유롭게 가입 및 탈퇴할 수 있으며 최대 10만명의 회원을 지원하고 메시지 기록을 저장합니다. 그룹에 가입하려면 사용자는 그룹 ID를 검색하고 신청서를 보내야 하며, 신청서는 관리자의 승인을 받지 않아도 그룹에 가입할 수 있습니다.

다음은 필요에 따라 게임에 적용할 수 있는 IM 그룹 특성을 기반으로 하는 몇 가지 샘플 솔루션을 제공합니다.

유형	솔루션	특성
채널 및 라이브 게임 로비	커뮤니티 그룹(Community)	커뮤니티 그룹은 다수의 사용자를 지원하며 사용자가 승인 없이 자유롭게 가입 및 탈퇴할 수 있도록 합니다.
친구	1:1 채팅+권한 제어(친구끼리만 메시지보내기)	친구끼리만 서로에게 메시지를 보낼 수 있습니다.
비공개 메시지	1:1 채팅+권한 제어(App 내 임의의 2명의 사용자가 서로 1:1 메시지를 보낼 수 있도록 허용)	두 명의 낯선 사람이 서로에게 메시지를 보낼 수 있습니다.
팀업	공개 그룹(Public) 및 회의 그룹(Meeting)	게임 내 팀원만 그룹 채팅에 입장할 수 있습니다. 오디오-비디오 채팅이 지원됩니다.
오디오-비디오 그룹	오디오-비디오 그룹(AVChatRoom)	그룹 구성원 수에는 제한이 없으며 사용자는 언제든지 오디오-비디오 그룹에 가입하거나 탈퇴할 수 있습니다.

설명：

친구 및 비공개 일대일 채팅 권한 제어에 대한 자세한 내용은 1:1 채팅 메시지 권한 제어를 참고하십시오.
그룹 시스템에 대한 자세한 내용은 그룹 시스템을 참고하십시오.