Tencent Real-Time Communication

Voice Chat Room (with UI)

Product Documentation

Contents

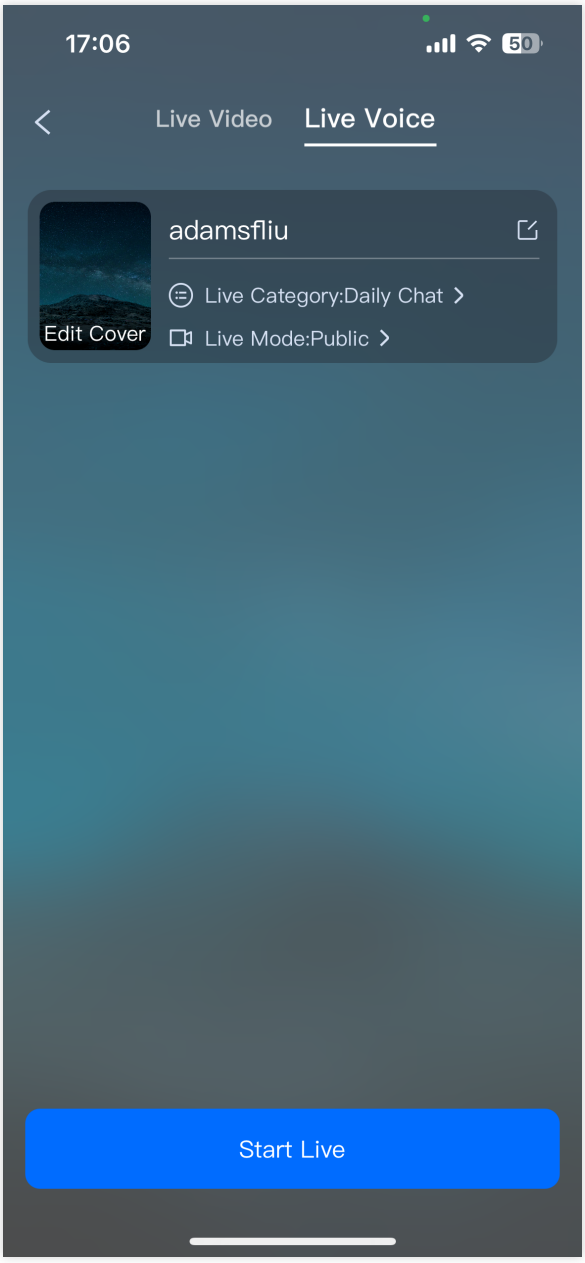
Voice Chat Room (with UI) Overview (TUILiveKit)

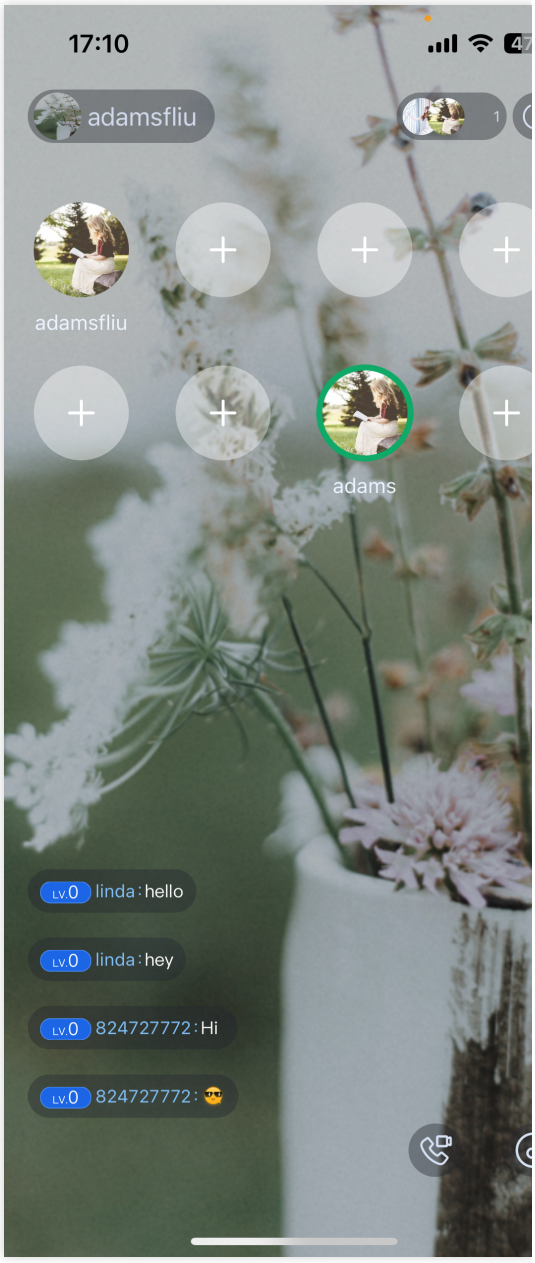
Last updated：2024-05-17 11:20:41

Overview

TUILiveKit is a product suitable for video interactive live streaming and voice chat scenarios, such as social entertainment, game interaction, real-time duet, voice radio, etc. By integrating this product, with just three steps and within 30 minutes, you can add features like voice chat rooms, interactive microphone connection, gift sending, room management, microphone position management, etc., to your application and quickly launch your service. The basic feature showcase is as follows:

Voice Live Preview Page	Voice Live Host Page





Supported Platform

Platform	Android	iOS
Supported		
Supported Languages/Frameworks	Java Kotlin	Swift Objective-C

Features

Basic Feature	Advanced Feature	Feature Advantages
HD Live Streaming Voice Chat Room Live Viewing Viewer Mic Connection View Viewer List Member Management Seat Management	Chat Barrage Heartbeat Like Interactive Gifts (Fullscreen Gifts) Sound Effects & Voice Changer	**Comprehensive UI Interaction** **Professional Live Streaming 3A Algorithm, Better Audio Quality** **Rich REST API**

Trying It Online

Platform	Android	iOS
Demo Integration	Github: Andorid	Github: iOS

Exchange and Feedback

If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

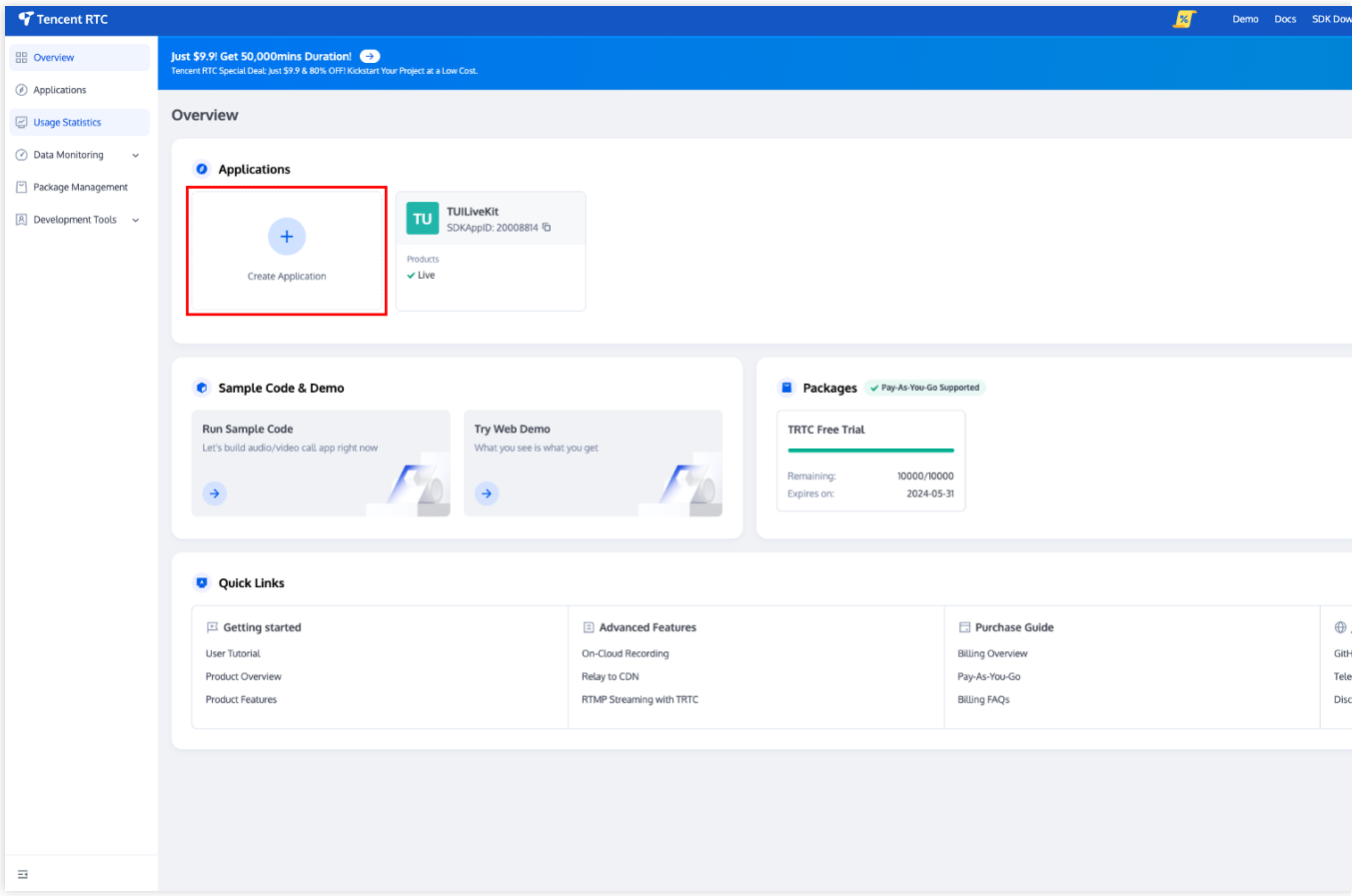
Activating the Service (TUILiveKit)

Last updated：2024-07-03 17:01:14

Activate Trial Service

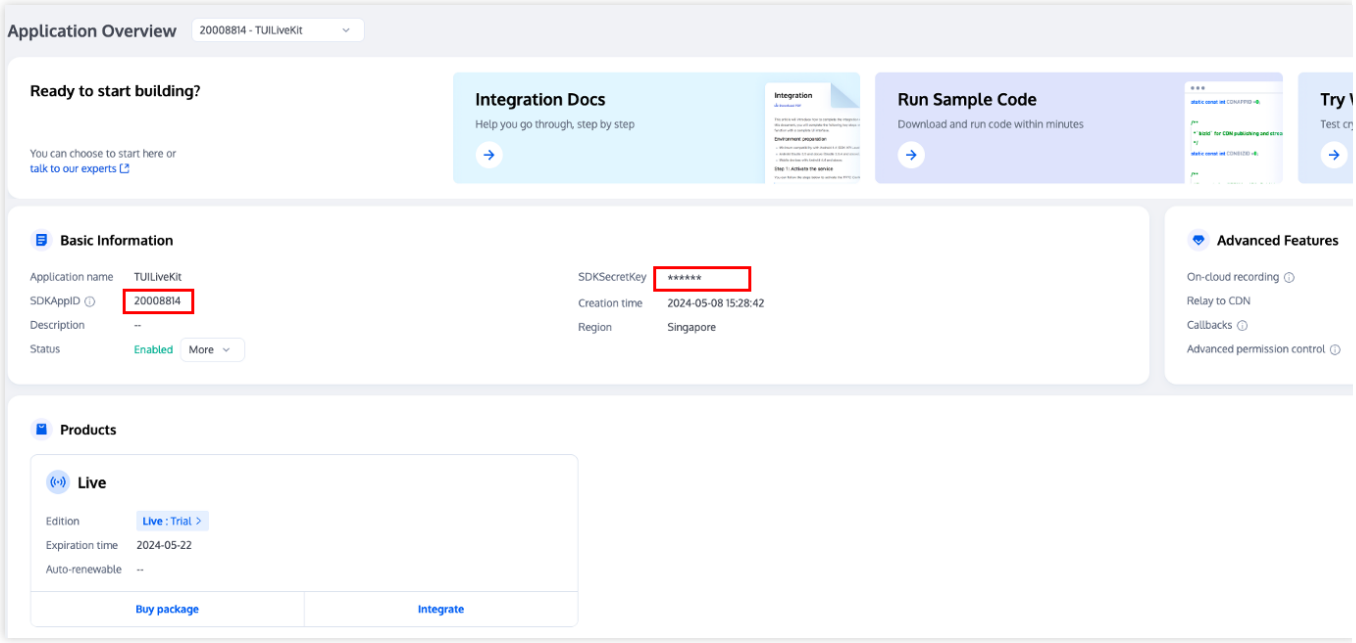
To give you a better experience with Interactive Live Streaming features, we provide a 14-day trial version for each SDKAppID for free (the trial version does not include additional call duration). Each SDKAppID can be tried for free twice, with each trial period lasting 14 days. Additionally, the total number of trials for all SDKAppIDs under one account is 10 times.

1. Log in to the Tencent RTC console, and click **Create Application.**



2. In the creation popup, enter **Application Name**, select **Live,** and choose the appropriate **Region**, then click **Create** to complete the creation of the trial version.

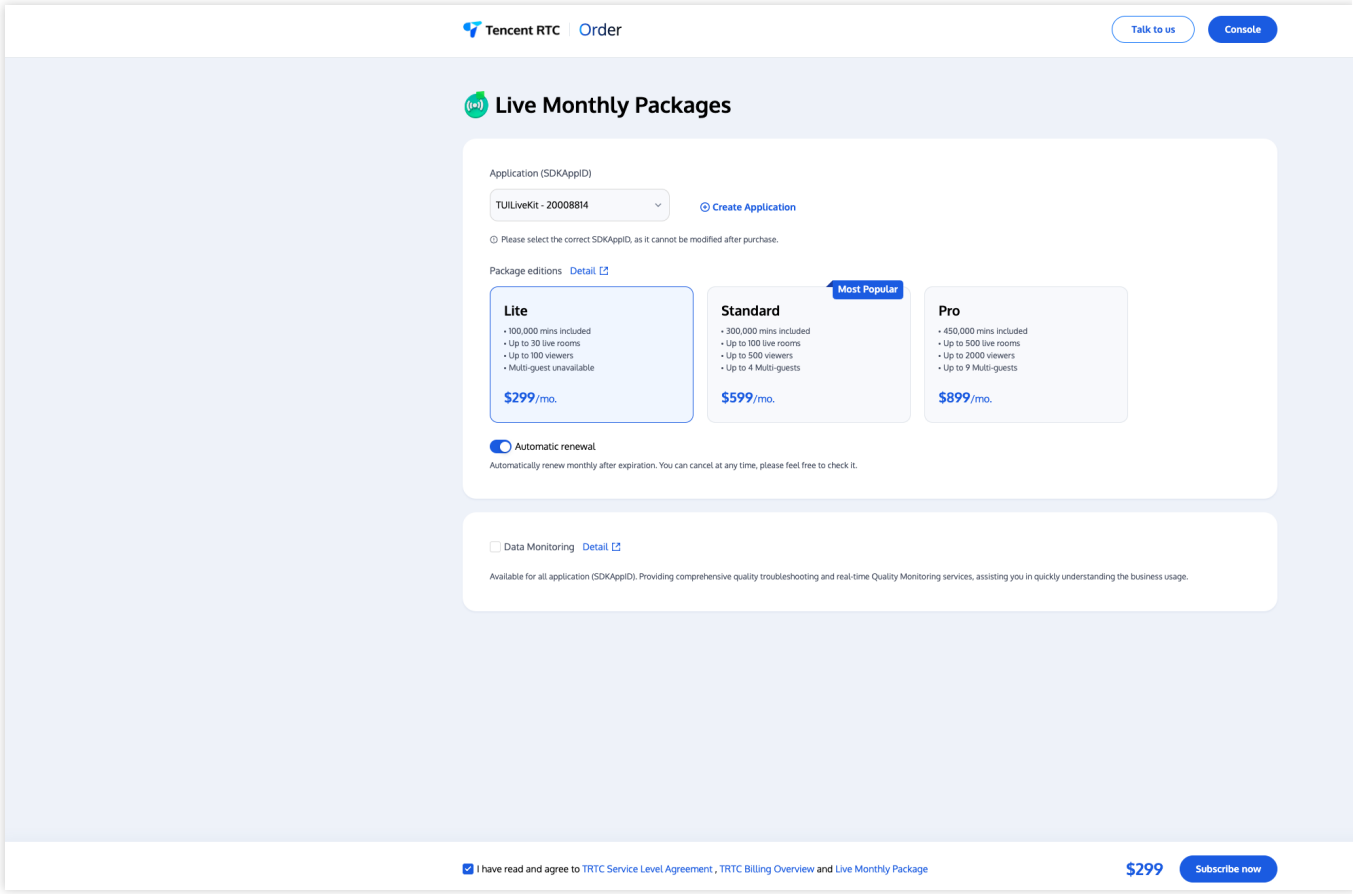
3. After creation, you can view detailed information about your app on the current page, such as `SDKAppID` , `SDKSecretKey` , and more. You can go to the Tencent RTC console to check application version information and refer to Quick Integration (TUILiveKit) for integration.



Purchasing the official version

Before officially going live, you need to purchase the official version of the Live Monthly Subscription Package. For price comparison and feature details of the Live Monthly Subscription Package, please refer to Tecnent RTC Live Monthly Packages. The package purchase process is as follows:

1. Visit the Live Purchase Page, select the application (SDKAppID) and version you wish to purchase, **and it is recommended to enable auto-renewal to avoid affecting your business usage**. After confirming the purchase information, check the agreement and click **Subscribe now.**



2. Go to the Order Confirmation Page to confirm product information and complete the payment.

Integration (TUILiveKit)
iOS

Last updated：2024-08-09 22:25:01

This article will guide you through the process of integrating the TUILiveKit component in a short time. Following this document, you will complete several key steps within an hour and ultimately obtain a video or voice live streaming feature with a complete UI interface.

Environment preparations

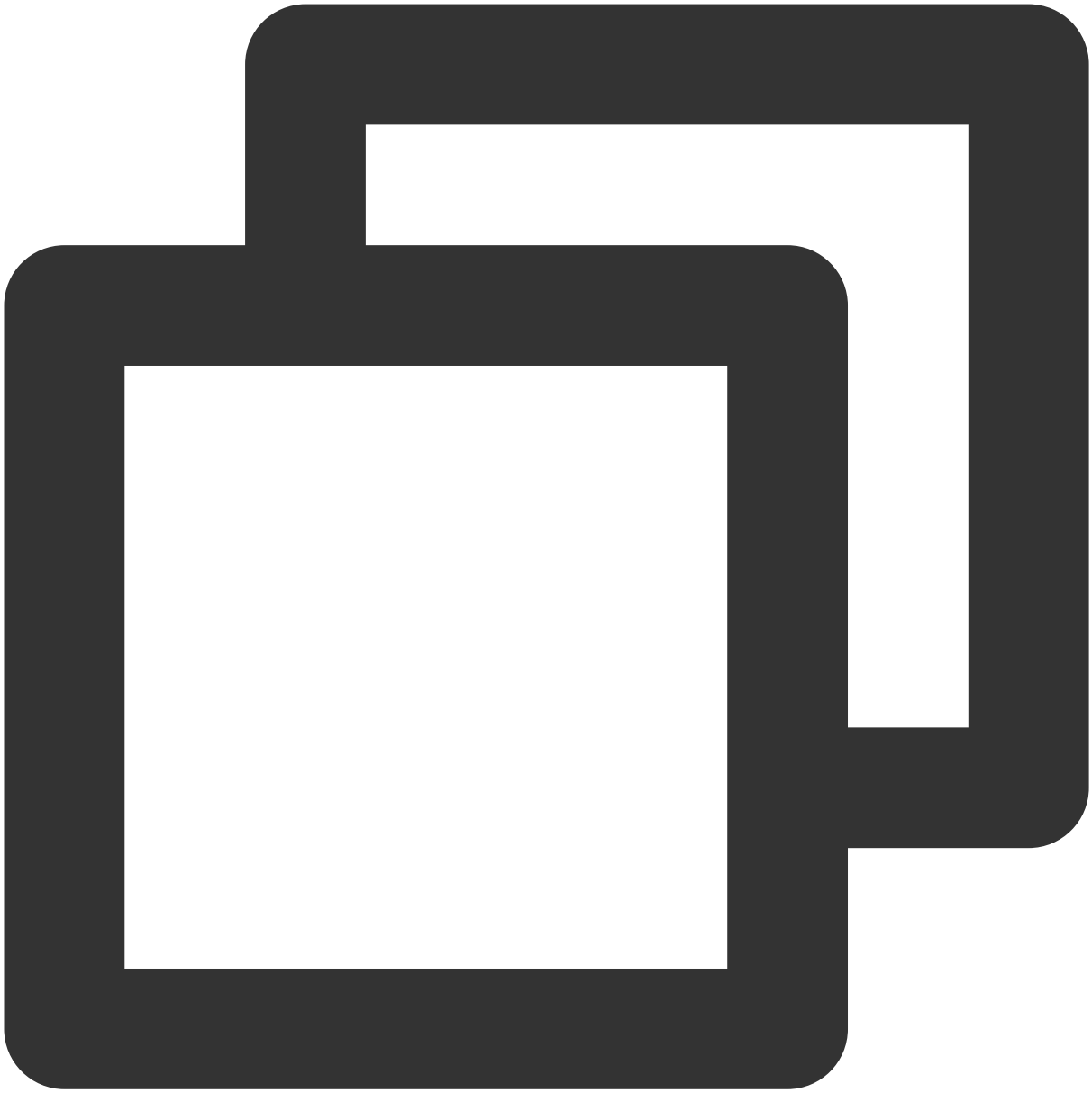
Xcode 15 or later
iOS 13.0 or later
CocoaPods environment installation, click to view.
If you encounter problems with access and use, see Q&A。

Step 1. Activate the service

Before using Tencent Cloud's audio and video services, you need to go to the console and activate the audio and video services for your application. For details, see Activate the Service (TUILiveKit).

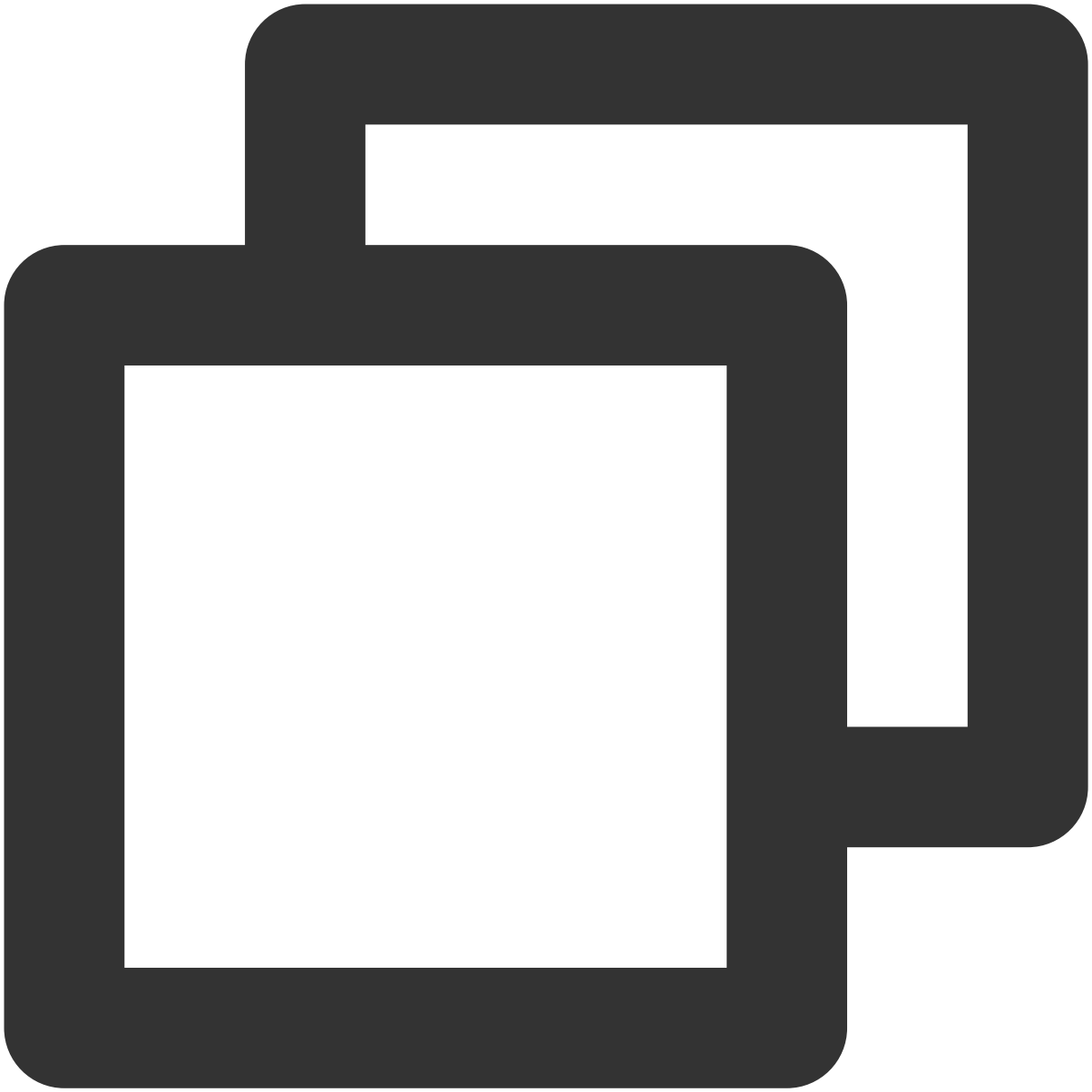
Step 2. Import the Component

Import components into CocoaPods. If problems exist, Please refer first Environment Preparation. The import components are as follows:
1. Please add the `pod 'TUILiveKit'` dependency to your `Podfile` file and refer to the Example project if you run into any problems.



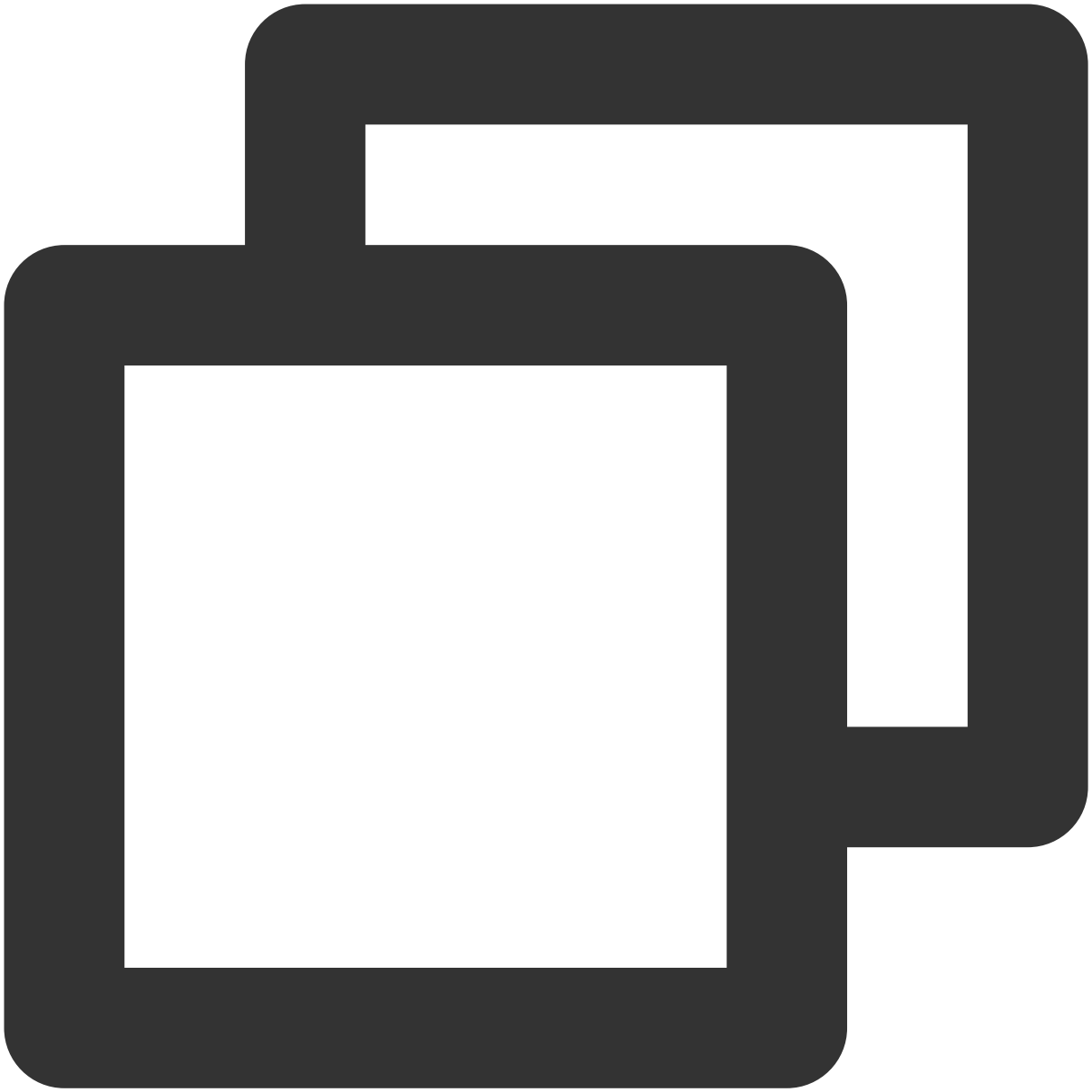
```
target 'xxxx' do
  ...
  ...
  pod 'TUILiveKit'
end
```

If you don't have a `Podfile` file, first terminal `cd` into the `xxxx.xcodeproj` directory and then create it with the following command:



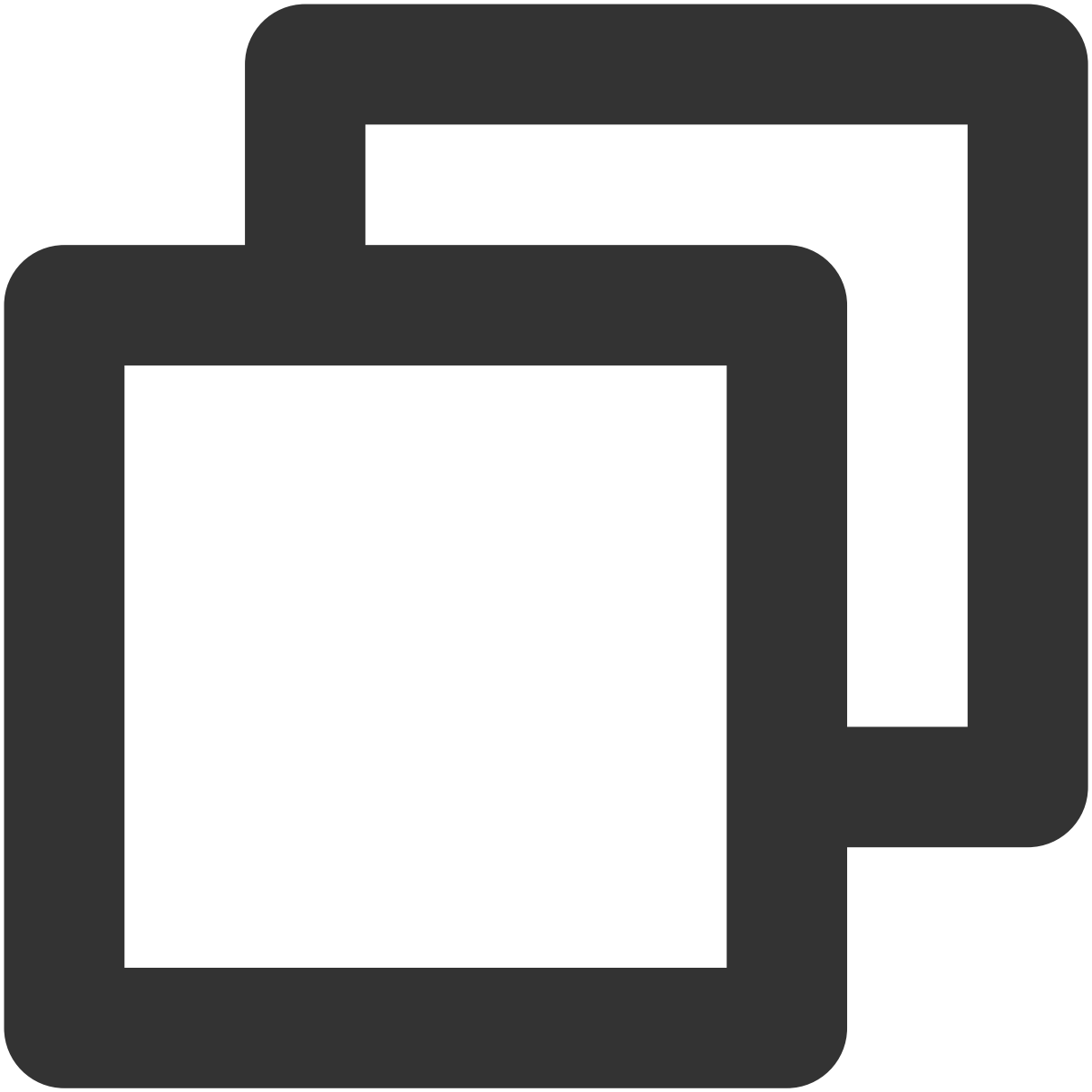
```
pod init
```

2. In the terminal, first `cd` to the `Podfile` directory, and then run the following command to install the component.



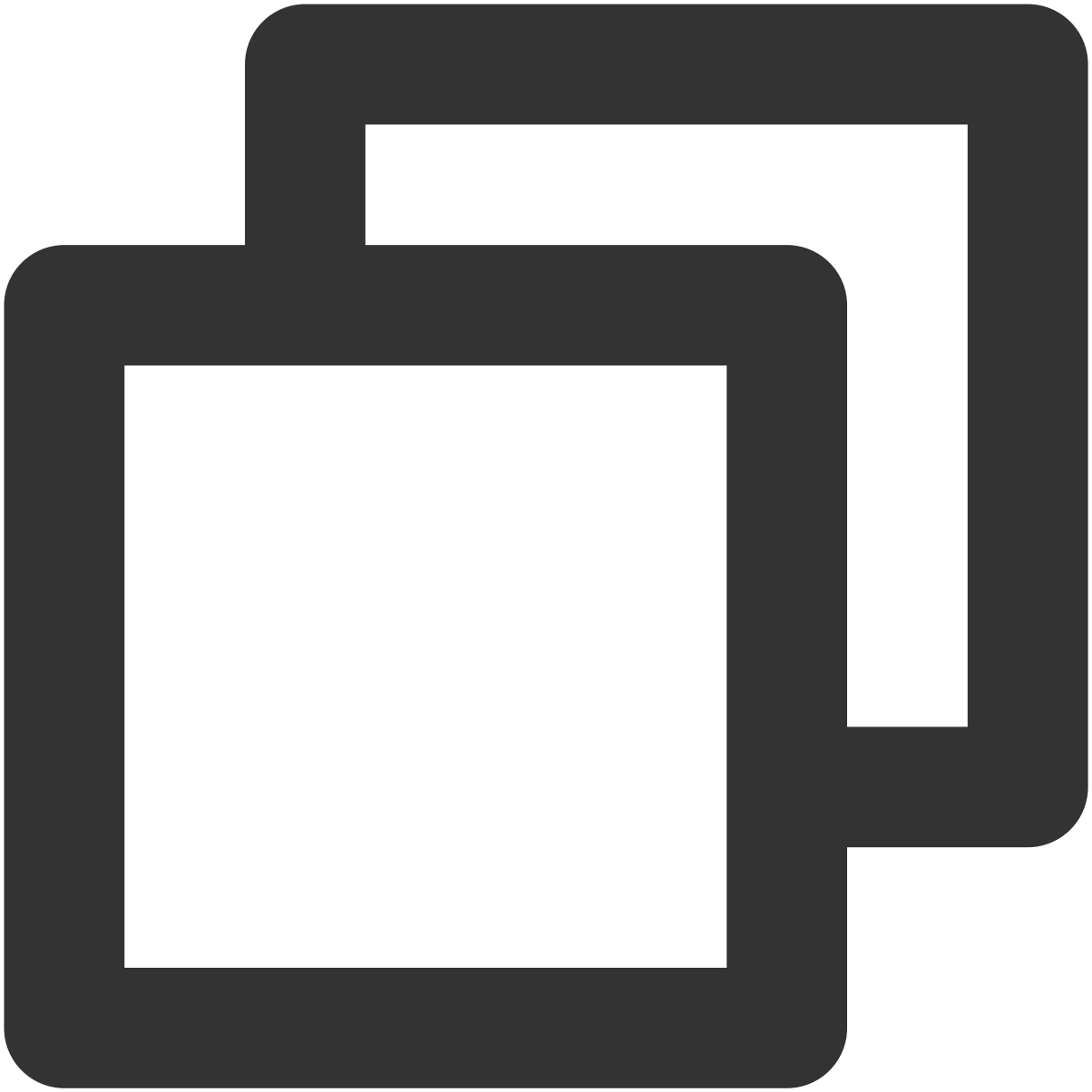
```
pod install
```

If the latest version of `TUILiveKit` cannot be installed, You can delete **Podfile.lock** and **Pods** first。Then update the CocoaPods repository list locally by executing the following command.



```
pod repo update
```

Afterwards, execute the following command to update the Pod version of the component library.

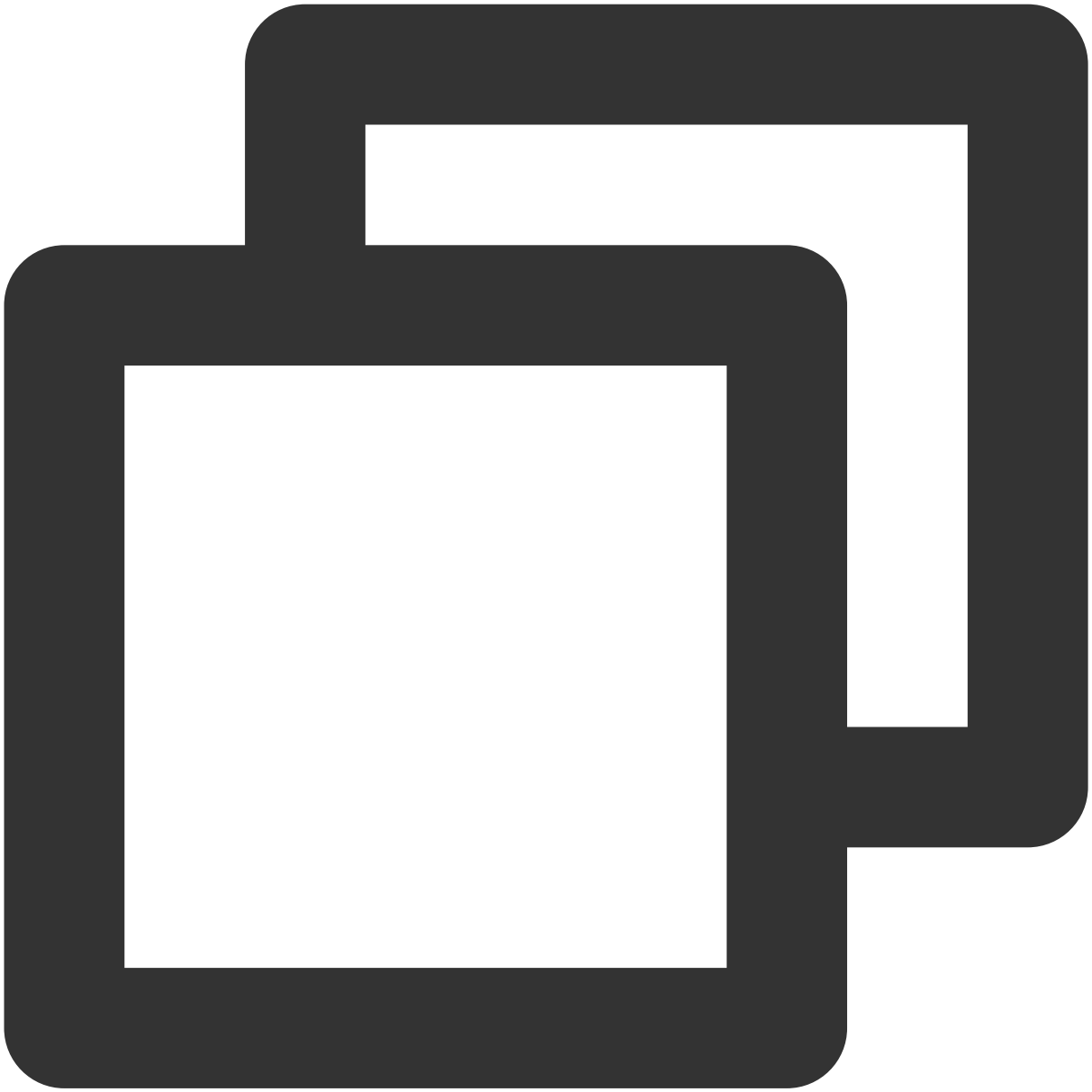


```
pod update
```

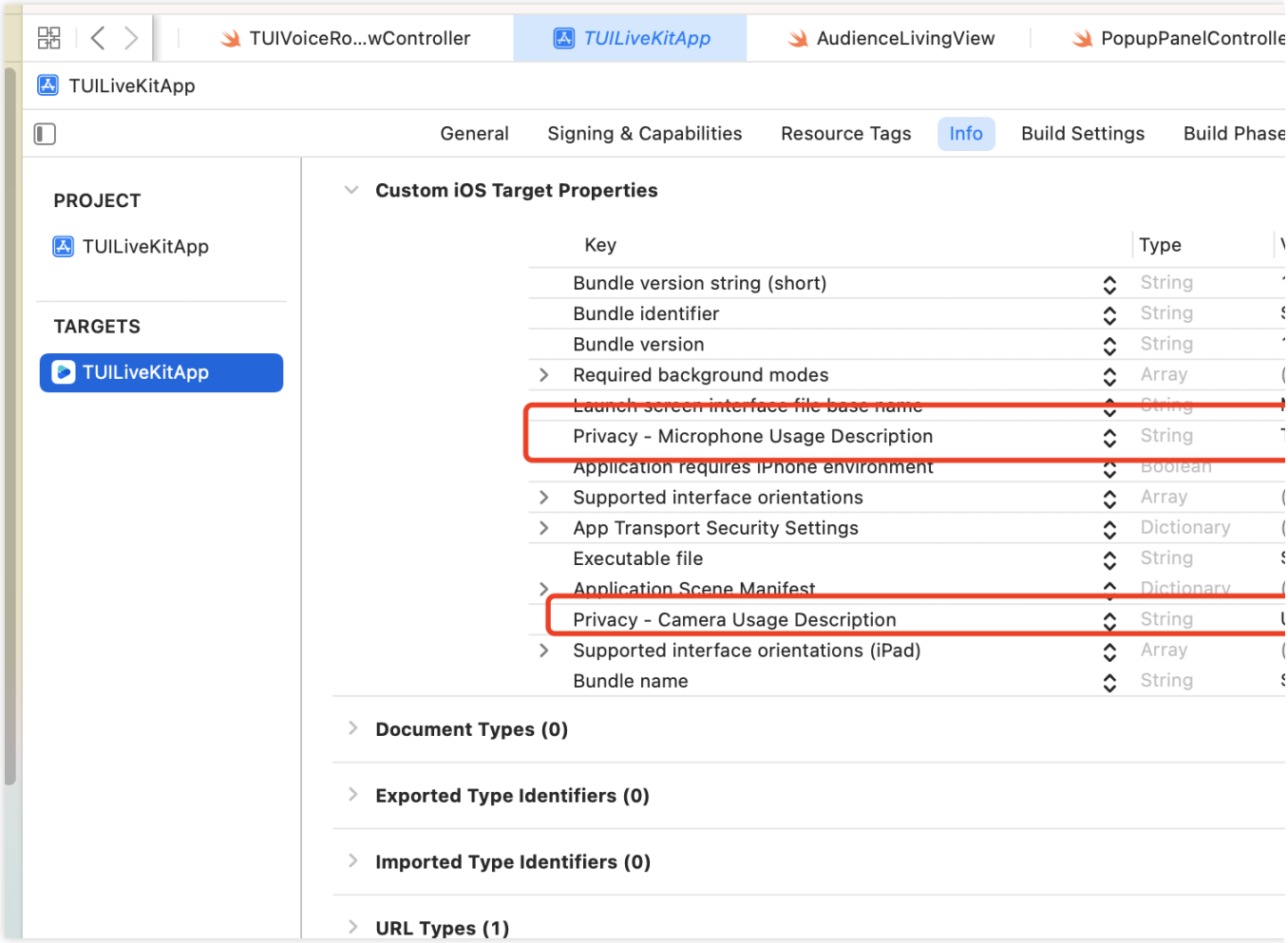
3. You can compile and run it first. If you run into problems, see Q&A. If the problem still can't be solved, you can run our Example project first. Any problems you encounter in the process of access and use, welcome to give us feedback.

Step 3. Configure the Project

To use audio and video features, microphone and camera usage permissions are required. In the App's Info.plist, add the following two items, corresponding to the prompt messages for microphone and camera when the system pops up the authorization dialog.



```
<key>NSCameraUsageDescription</key>
<string>TUILiveKit needs access to your camera to capture video.</string>
<key>NSMicrophoneUsageDescription</key>
<string>TUILiveKit needs access to your mic to capture audio.</string>
```

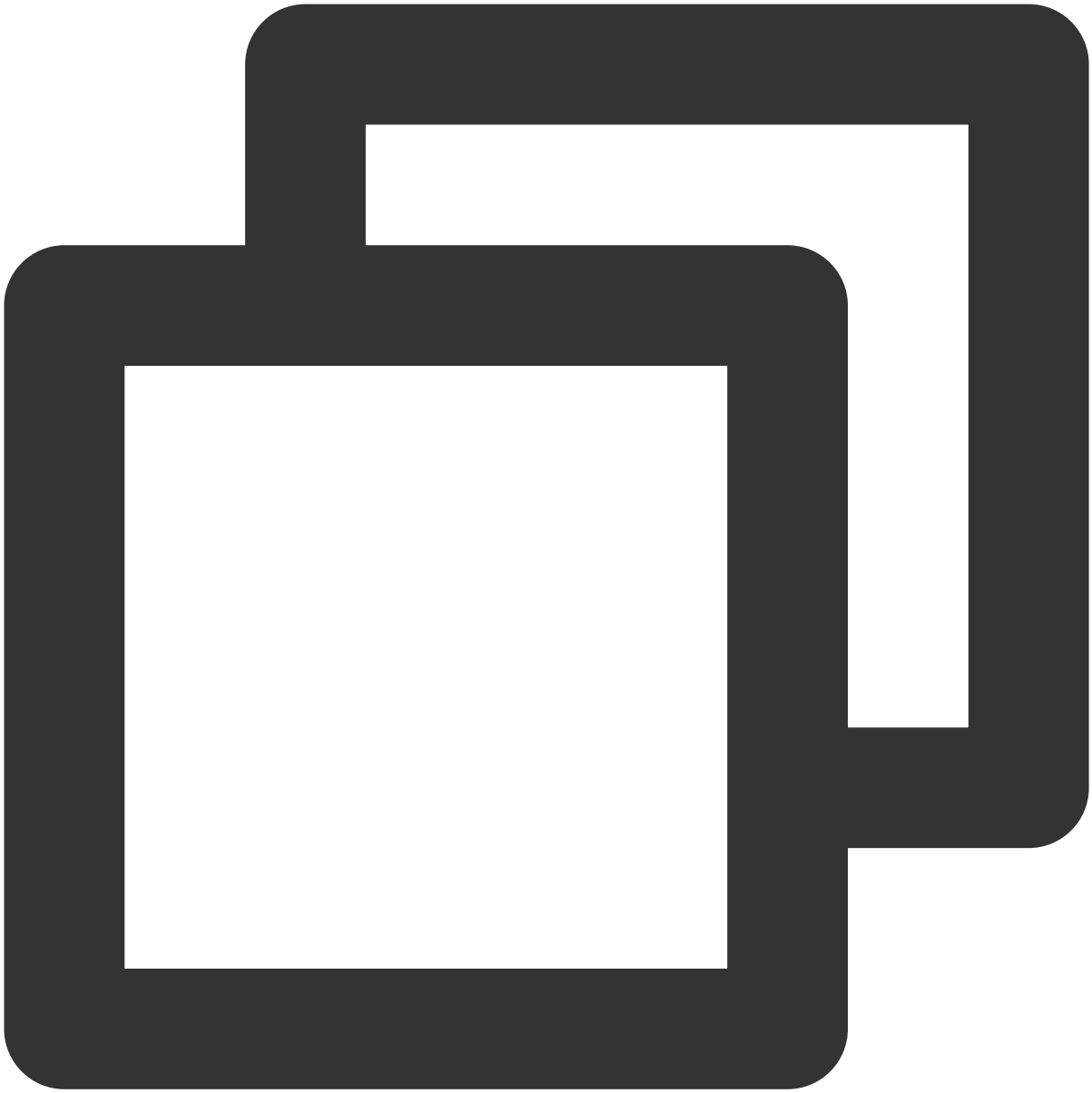


Step 4. Log in

Add the following code to your project, which completes the login of the TUI component by calling the relevant interface in TUICore. This step is very important, because you can use all functions of TUILiveKit only after the login is successful. Therefore, please patiently check whether the relevant parameters are correctly configured.

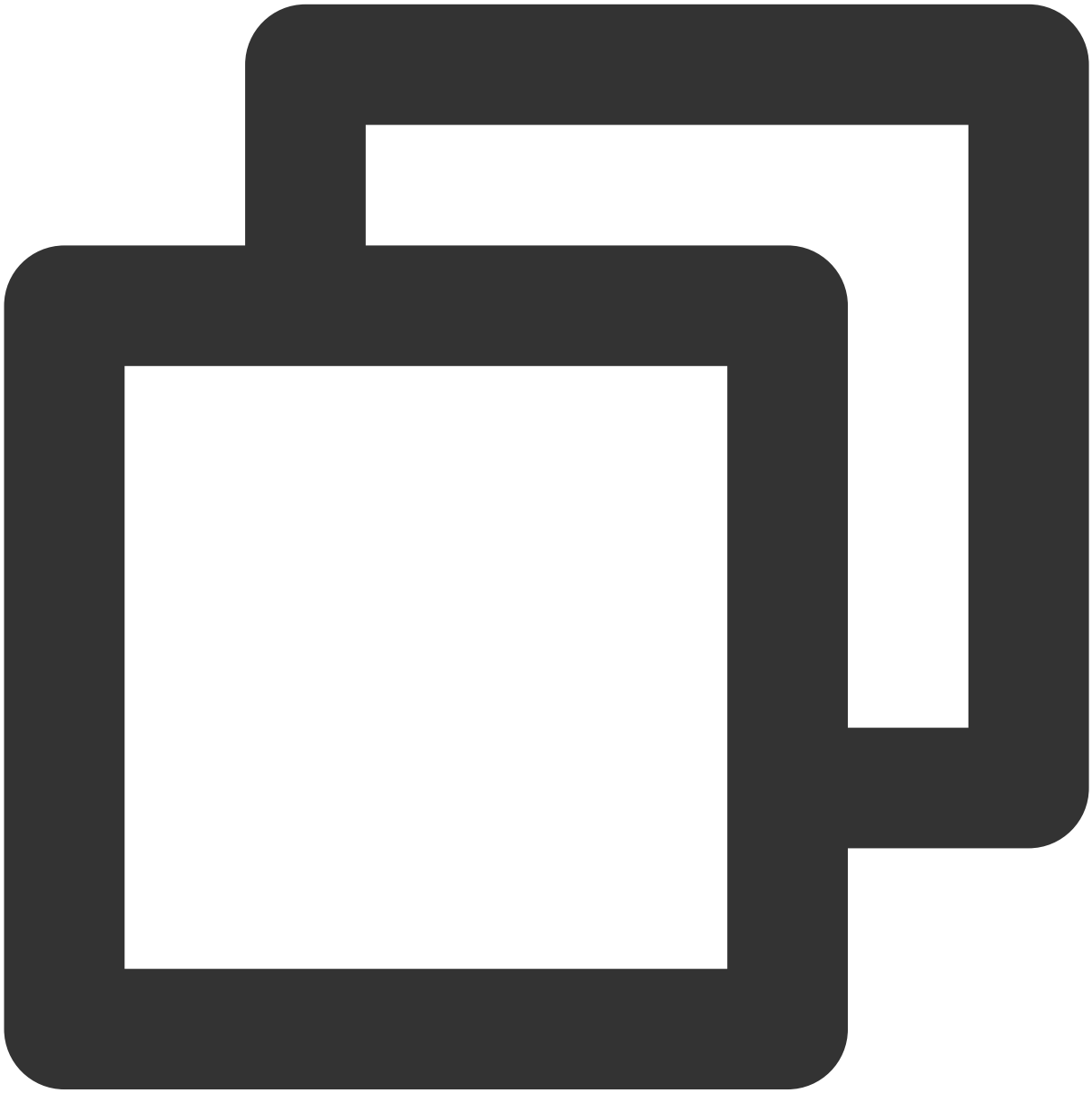
Swift

Objective-C



```
//
//  AppDelegate.swift
//

import TUICore
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch

    TUILogin.login(1400000001,                // Replace it with the SDKAppID obtai
            userID: "denny",                 // Please replace it with your UserID
            userSig: "xxxxxxxxxxx") {        // You can calculate a UserSig in the
      print("login success")
```

```
    } fail: { (code, message) in
      print("login failed, code: \\(code), error: \\(message ?? "nil")")
    }

    return true
}
```



```
//
//  AppDelegate.m
//
```

```
#import <TUICore/TUILogin.h>
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    [TUILogin login:1400000001        // Replace it with the SDKAppID obtained in S
             userID:@"denny"          // Please replace it with your UserID
            userSig:@"xxxxxxxxxxx"    // You can calculate a UserSig in the console
               succ:^{
        NSLog(@"login success");
    } fail:^(int code, NSString * _Nullable msg) {
        NSLog(@"login failed, code: %d, error: %@", code, msg);
    }];
    return YES;
}
```

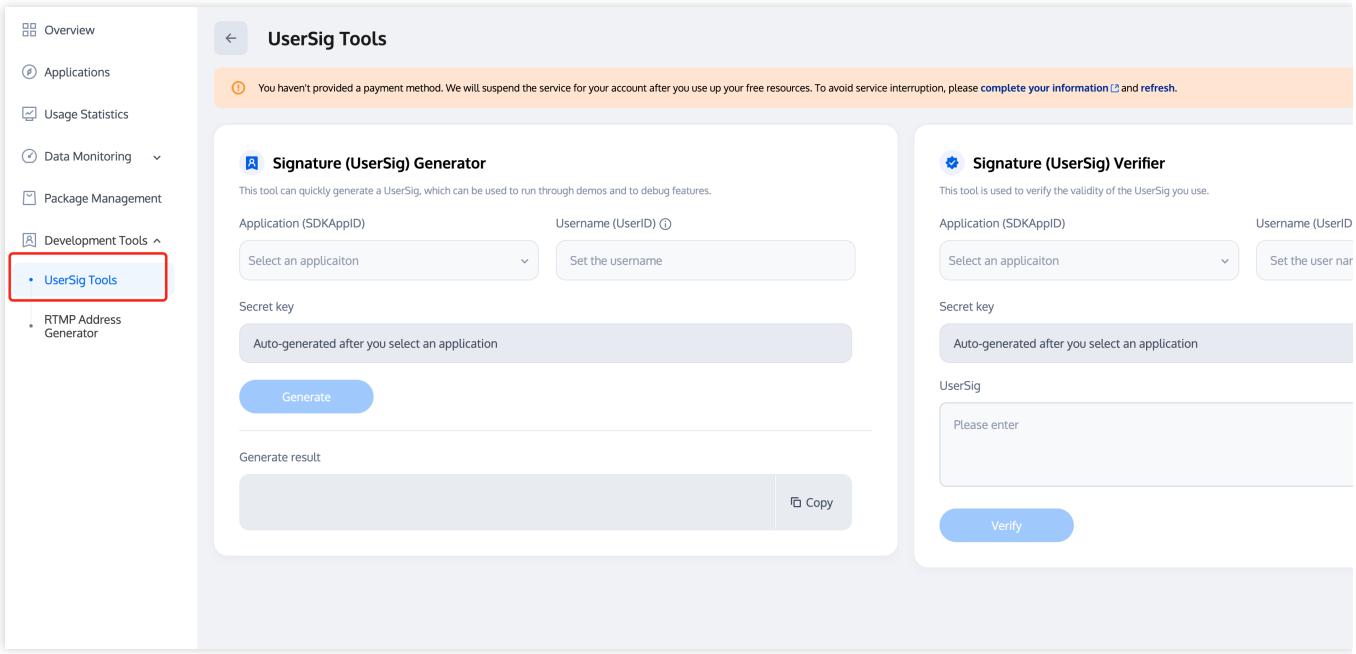
Parameter description

This section details the key parameters required in the login function:

SDKAppID: Obtained in the last step in Step 1 and not detailed here.

UserID: The ID of the current user, a string type, can only include letters (a–z and A–Z), digits (0–9), hyphens (-), and underscores (_).

UserSig: The authentication credential used by Tencent Cloud to verify whether the current user is allowed to use the TRTC service. You can get it by using the `SDKSecretKey` to encrypt the information such as `SDKAppID` and `UserID`. You can generate a temporary `UserSig` by clicking the UserSig Generate button in the console.



For more information, see UserSig.

Note:

Many developers have contacted us with many questions regarding this step. Below are some of the frequently encountered problems:

SDKAppID is invalid.

userSig is set to the value of Secretkey mistakenly. The userSig is generated by using the SecretKey for the purpose of encrypting information such as sdkAppId, userId, and the expiration time. But the value of the userSig that is required cannot be directly substituted with the value of the SecretKey.

userId is set to a simple string such as 1, 123, or 111, and your colleague may be using the same userId while working on a project simultaneously. In this case, login will fail as TRTC doesn't support login on multiple terminals with the same UserID. Therefore, we recommend you use some distinguishable userId values during debugging.

The sample code on GitHub uses the `genTestUserSig` function to calculate `UserSig` locally, so as to help you complete the current integration process more quickly. However, this scheme exposes your `SecretKey` in the application code, which makes it difficult for you to upgrade and protect your `SecretKey` subsequently. Therefore, we strongly recommend you run the `UserSig` calculation logic on the server and make the application request the `UserSig` calculated in real time every time the application uses the `TUILiveKit` component from the server.

Step 5. Enter the Voice Chat Room Preview Screen

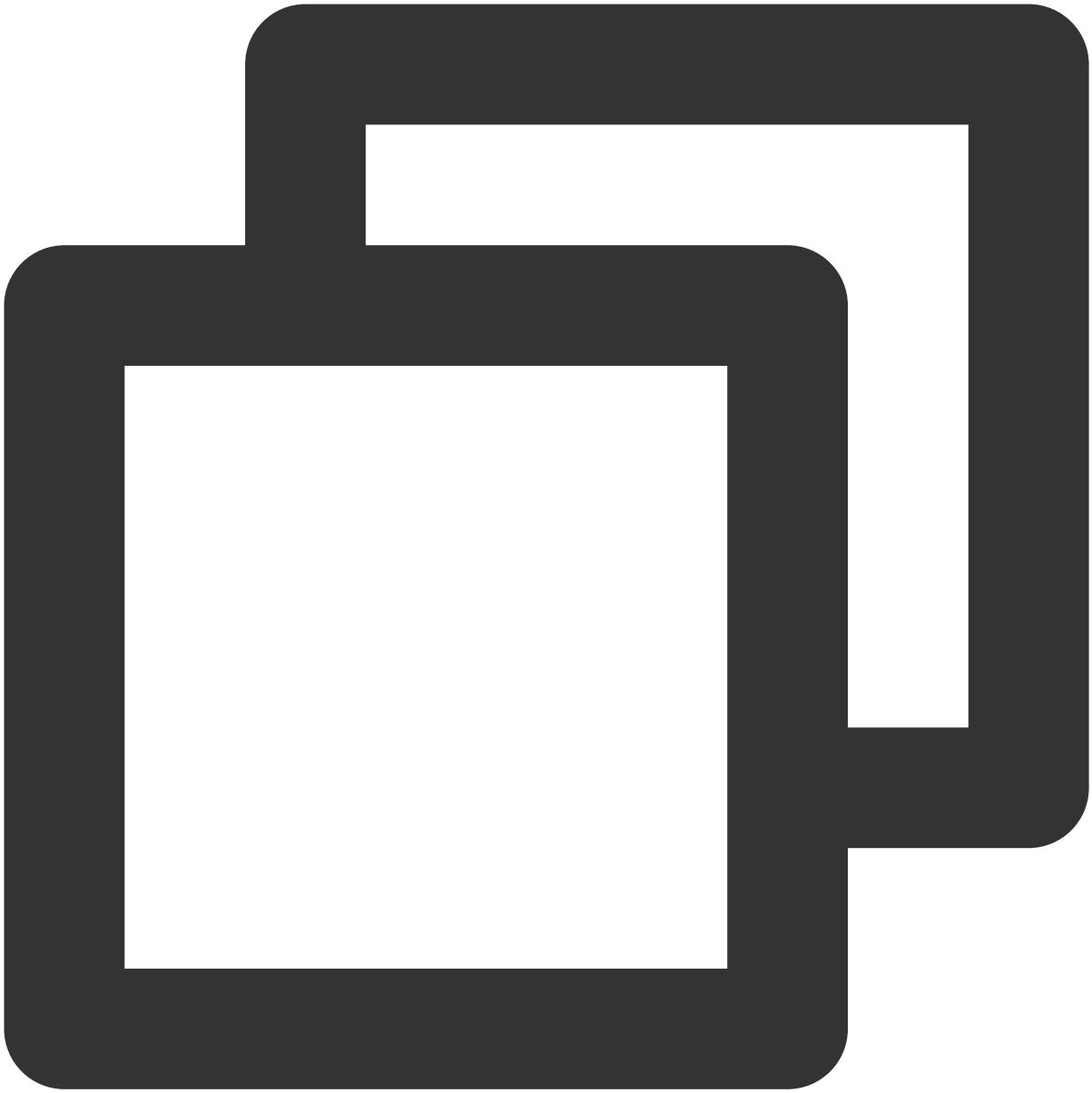
Note：

It's important to make sure you've followed Step 4 to complete the login. Only after you log in to TUILogin.login can you enter the live preview screen normally.

By loading the TUILiveKit's TUIVoiceRoomViewController page, you can launch the preview screen. Clicking "Start Live Streaming" will create a voice chat room.

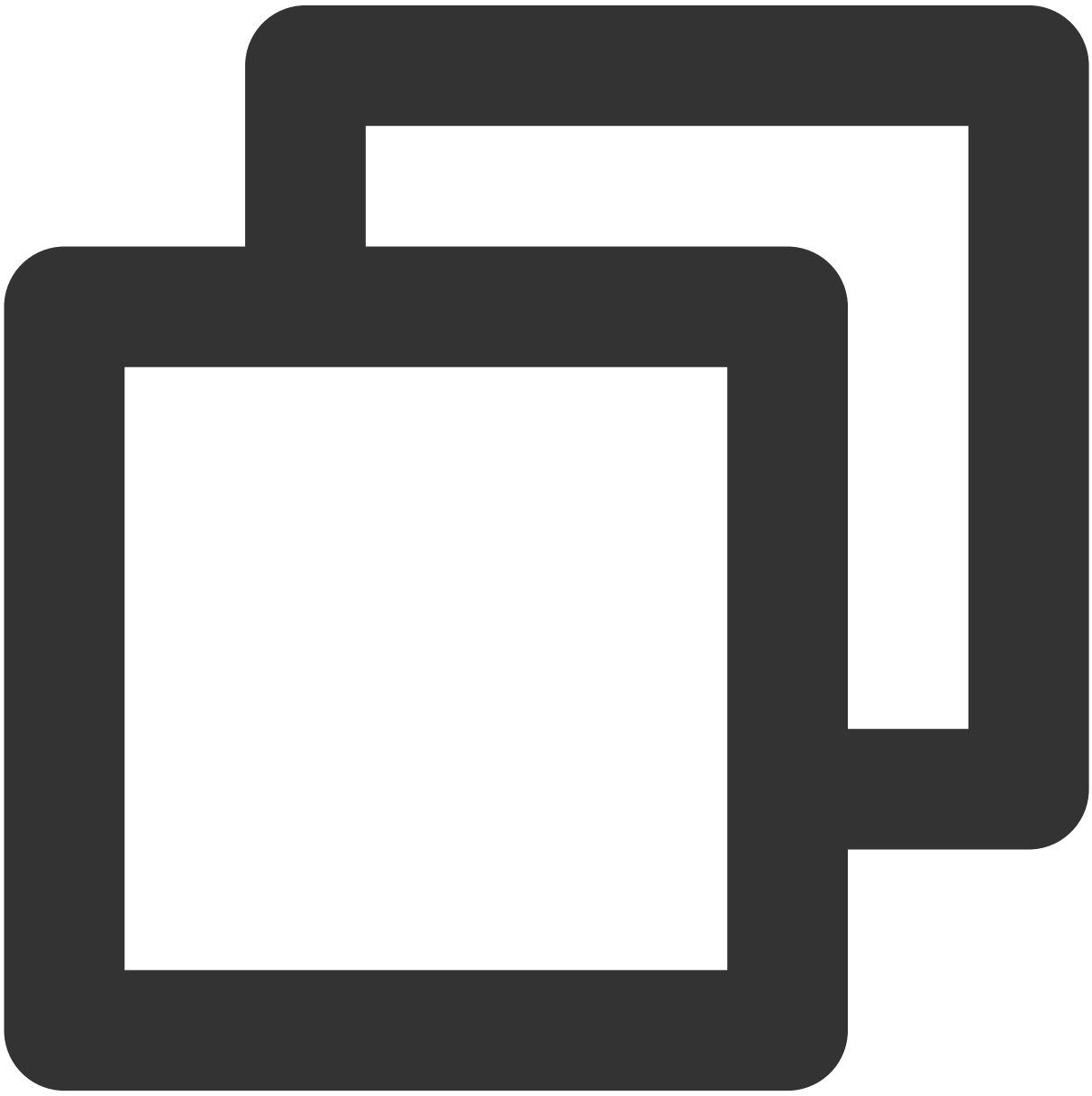
Swift

Objective-C



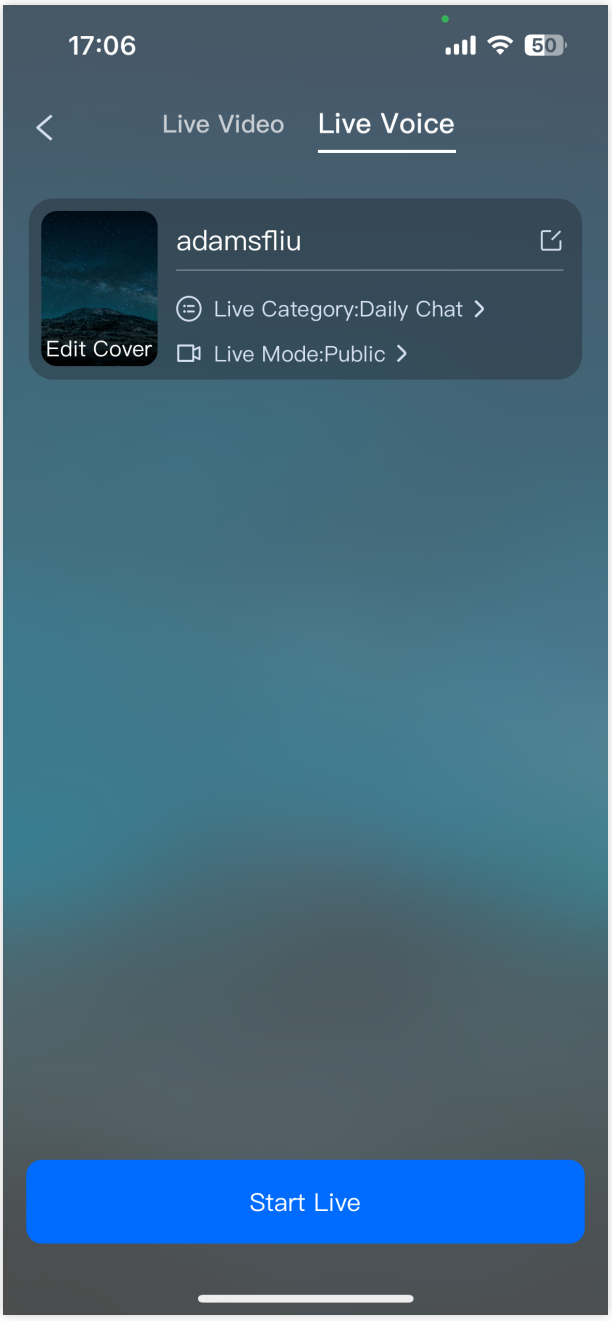
```
//
//  MainViewController.swift
//

import UIKit
import TUILiveKit

@objc private func buttonTapped(_ sender: UIButton) {
    // RoomId can be custom generated, Recommended use LiveIdentityGenerator.shared
    let roomId = "123666";
```

```
    // Enter the voice chat room preview screen
    var roomParams = RoomParams()
    //The default value is the maximum number of seat supported by the package
    roomParams.maxSeatCount = 0
    roomParams.seatMode = .applyToTake

    let voiceRoomController = TUIVoiceRoomViewController(roomId: roomId, behavior:
    self.navigationController?.pushViewController(voiceRoomController, animated: tr
}
```

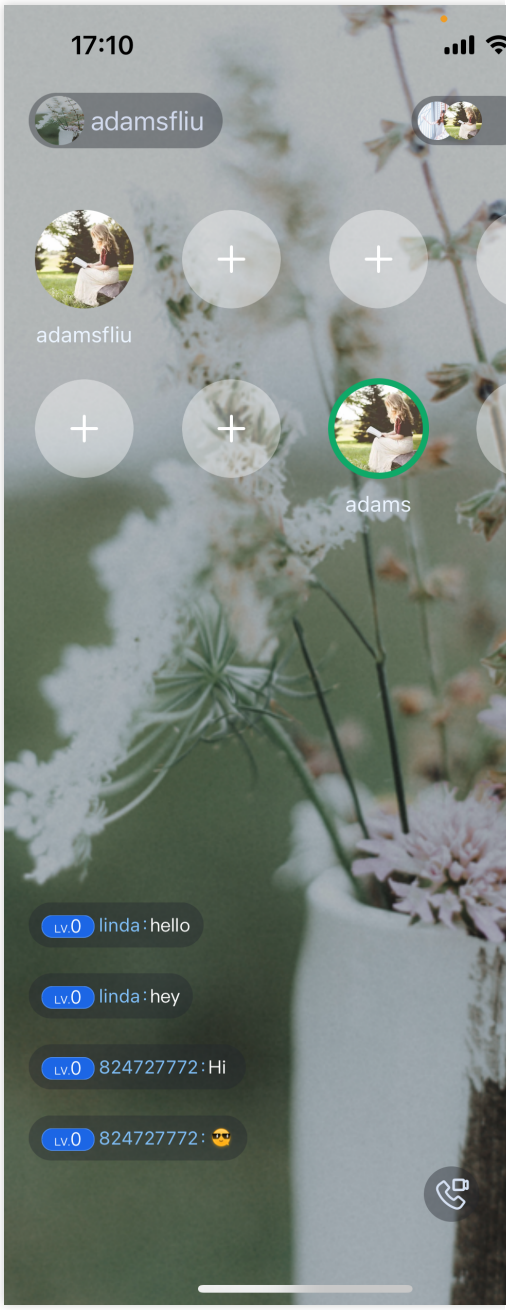


```
//
```

```
//  MainViewController.m
//

#import <TUILiveKit/TUILiveKit-Swift.h>
#import <RTCRoomEngine/RTCEngine.h>

- (void)buttonTapped:(UIButton *)sender {
    // RoomId can be custom generated, Recommended use [LiveIdentityGenerator.share
    NSString *roomId = @"123666";

    // Enter the voice chat room preview screen
    RoomParams *roomParams = [[RoomParams alloc] initWithMaxSeatCount:0  //The defa
                                                                seatMode:TUISeatModeAp
    TUIVoiceRoomViewController *voiceRoomController = [[TUIVoiceRoomViewController


    [self.navigationController pushViewController:voiceRoomController animated:true
}
```



Voice chat room preview screen



Voice chat room in-room screen

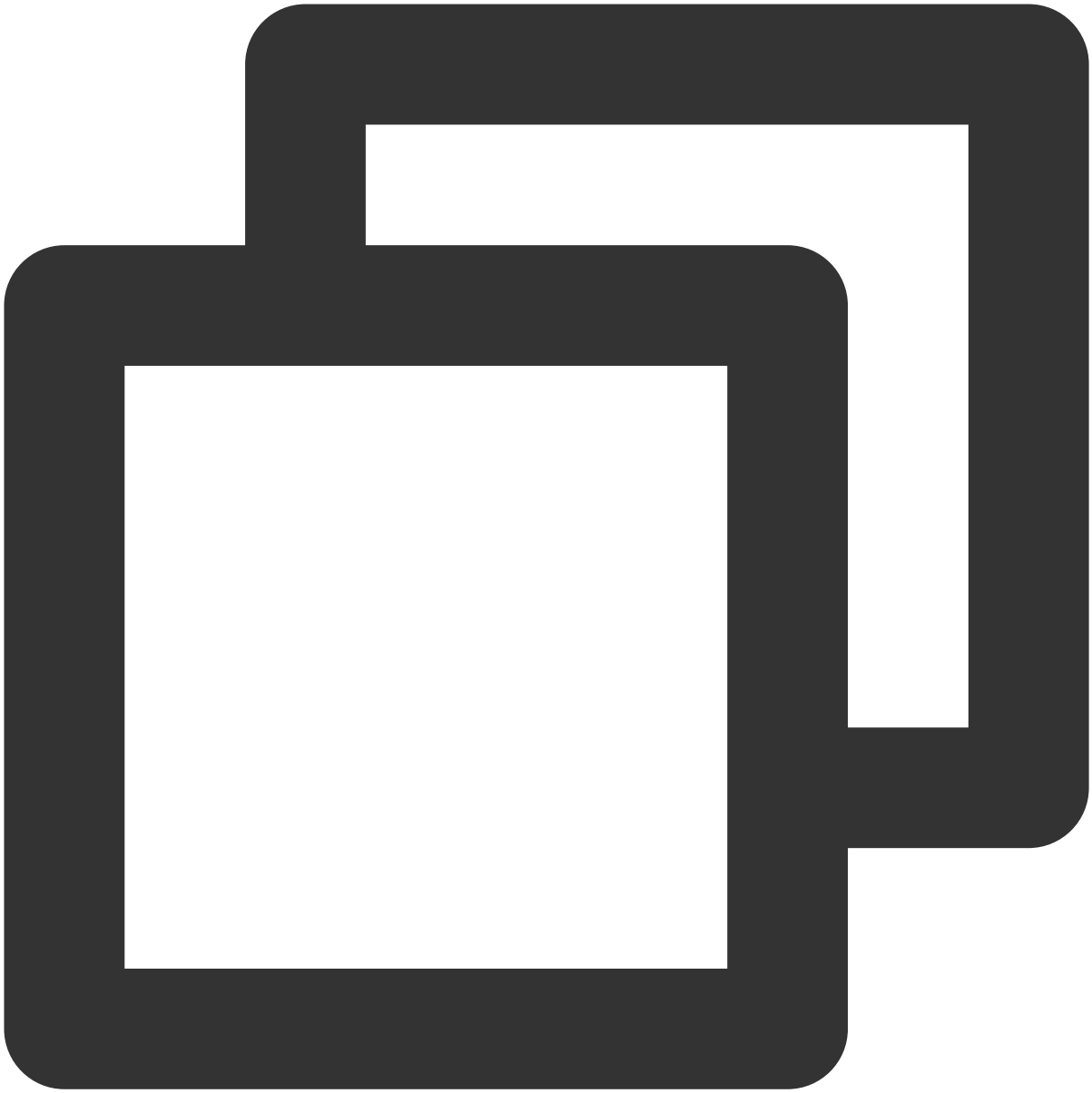
Step 6. Pull the room list

Note：

It's important to make sure you've followed Step 4 to complete the login. Only after you log in to TUILogin.login can you enter the live preview screen normally.

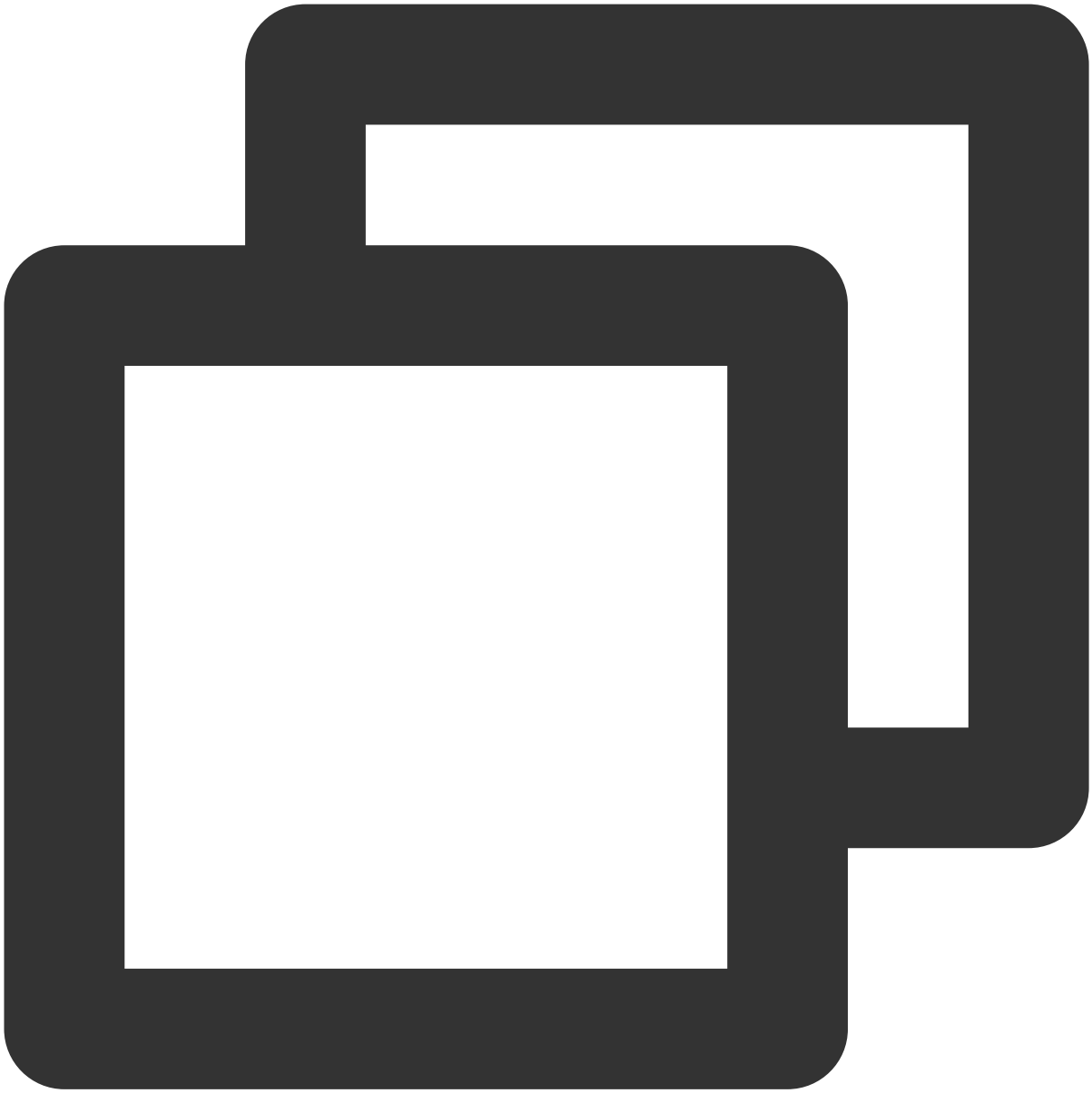
Swift

Objective-C



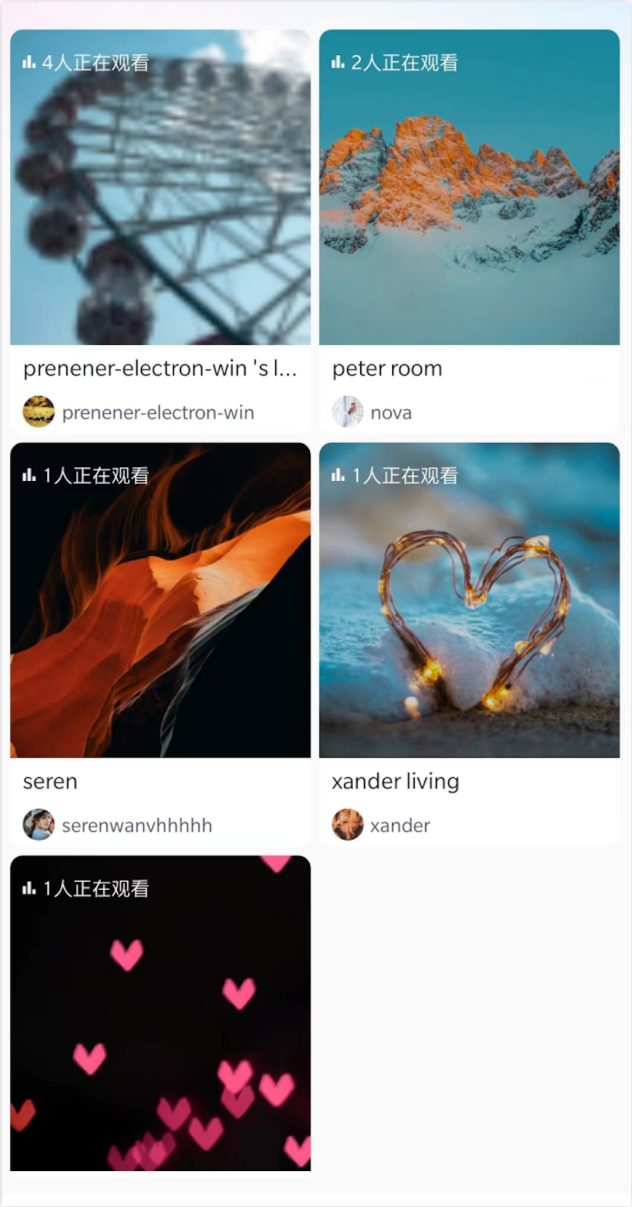
```
//
//  MainViewController.swift
//

import UIKit
import TUILiveKit

@objc private func buttonTapped(_ sender: UIButton) {
    // Enter room list
    let liveListViewController = TUILiveListViewController()
    self.navigationController?.pushViewController(viewController, animated: true)
```

```
}
```



```
//
//  MainViewController.m
//

#import <TUILiveKit/TUILiveKit-Swift.h>

- (void)buttonTapped:(UIButton *)sender {
    // Enter room list
    TUILiveListViewController *liveListViewController = [[TUILiveListViewController
```

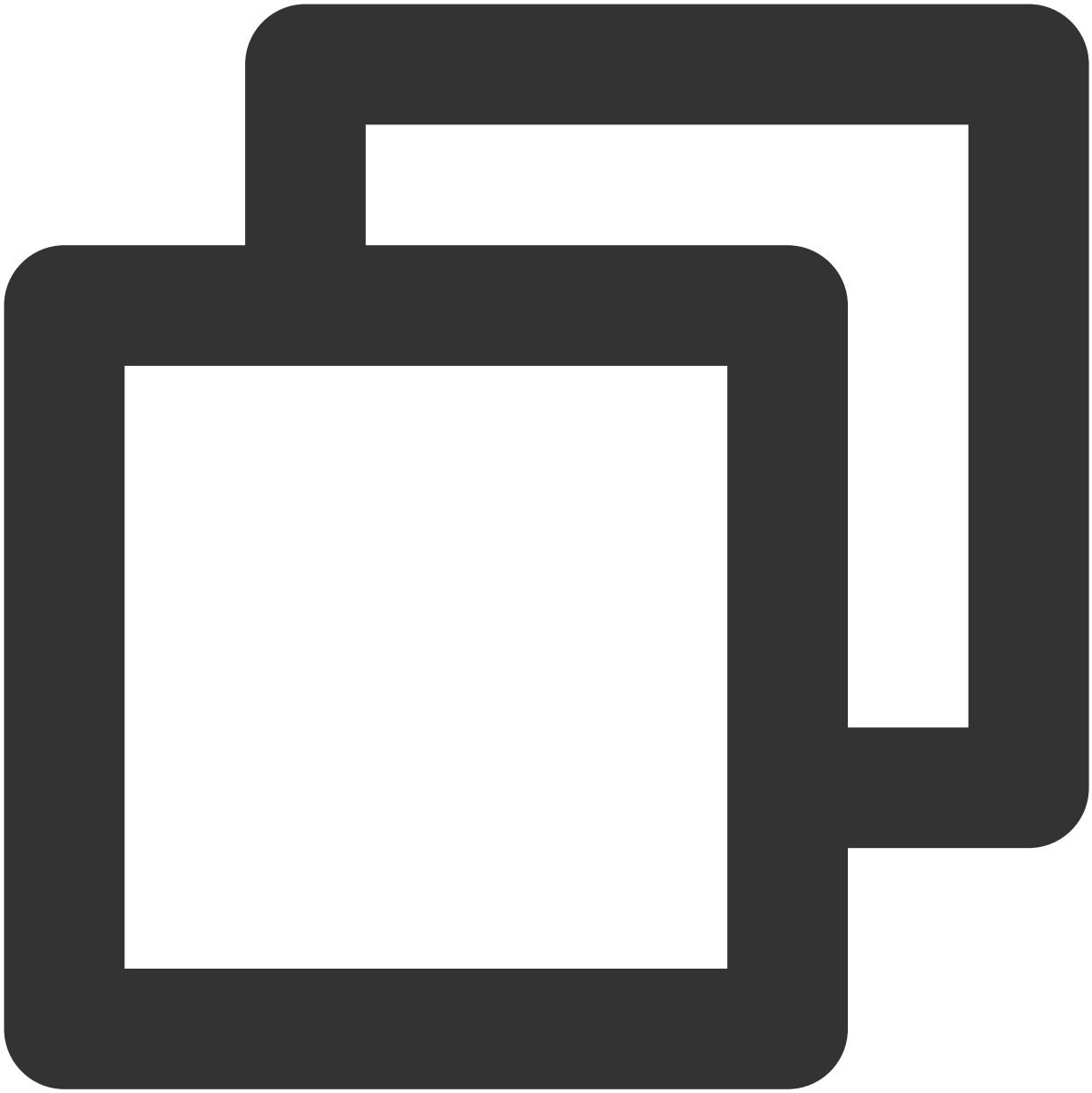
```
    [self.navigationController pushViewController:liveListViewController animated:t
}
```



Step 7. Audience enters the live streaming room

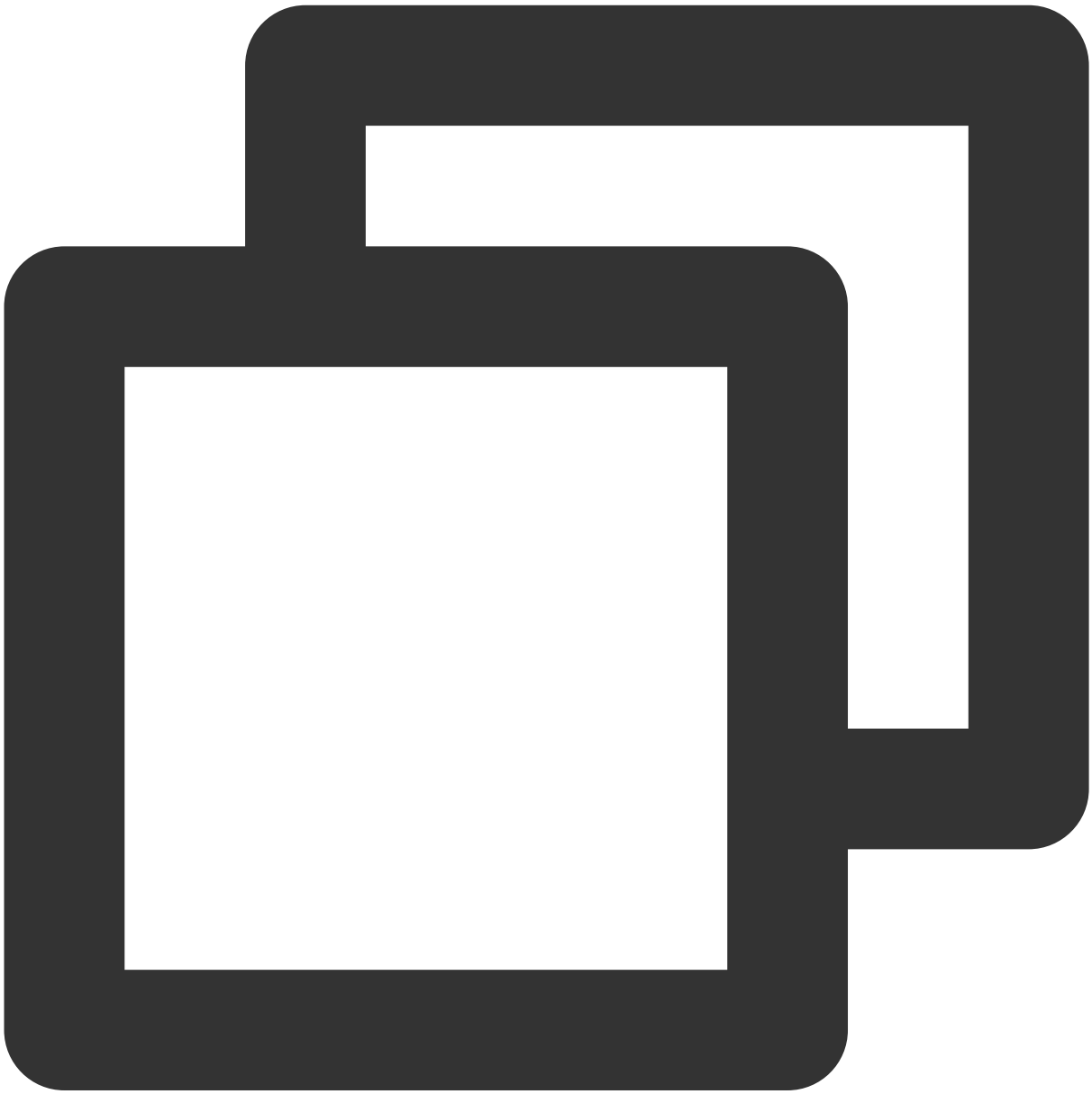
Swift

Objective-C



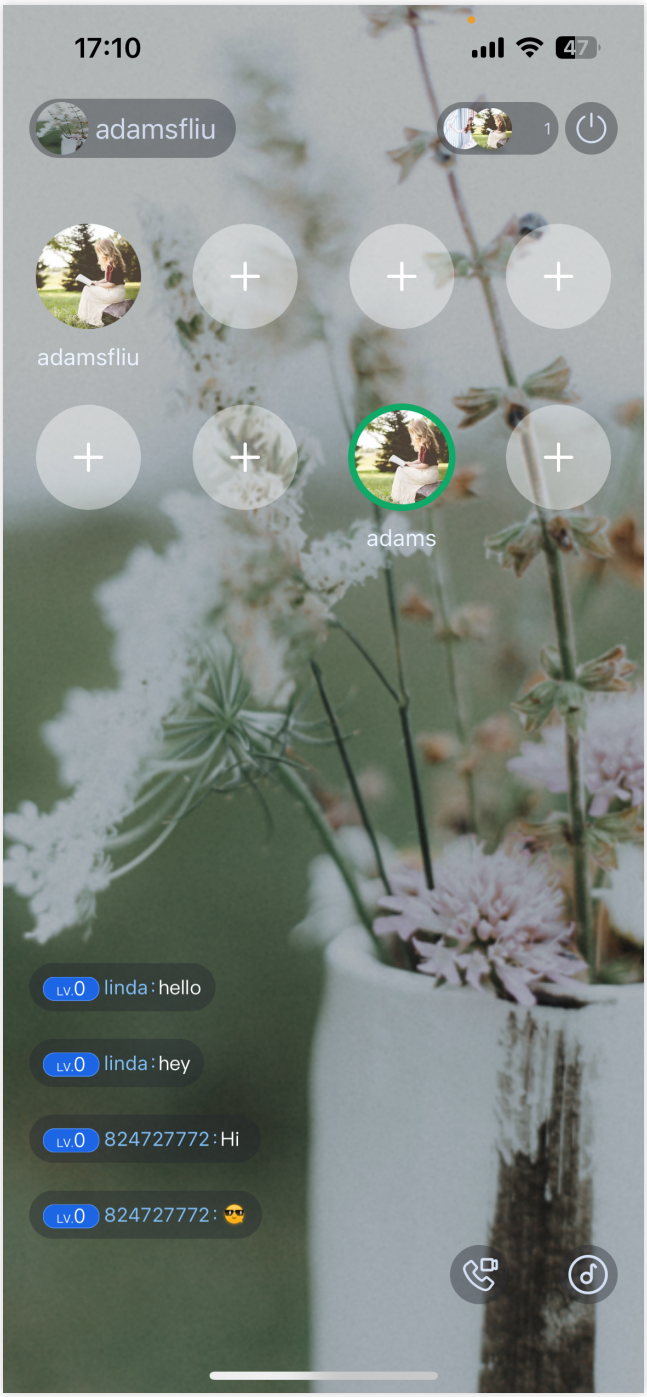
```
//
//  MainViewController.swift
//

import UIKit
import TUILiveKit

@objc private func buttonTapped(_ sender: UIButton) {
    // RoomId is the room ID for creating a voice chat room
    let roomId = "123666";
```

```
    // Enter the voice chat room
    let viewController = TUIVoiceRoomViewController(roomId: roomId, behavior: .join
    self.navigationController?.pushViewController(viewController, animated: true)
}
```

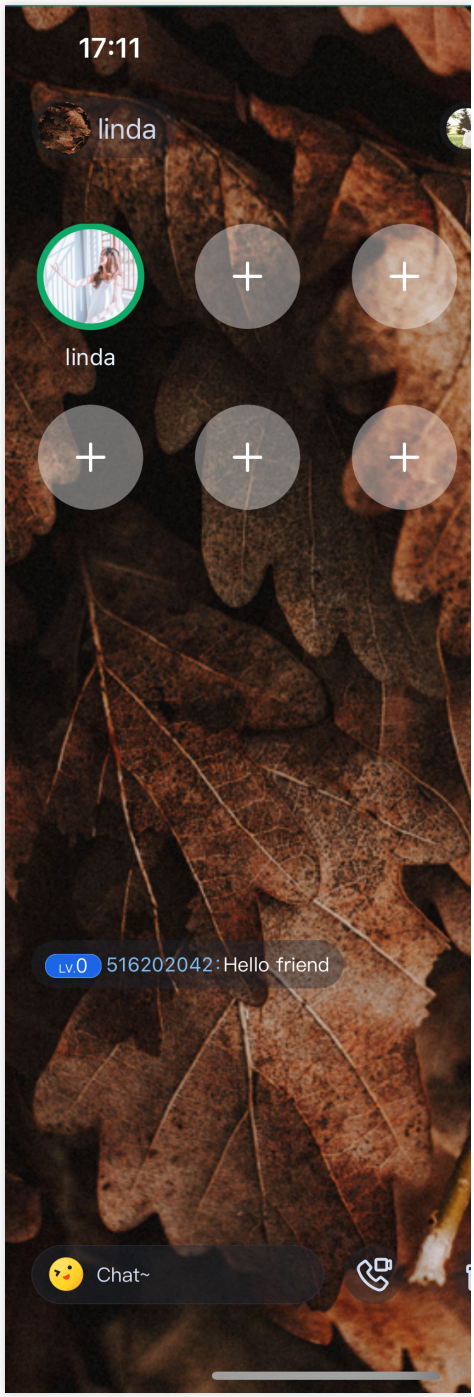


```
//
//  MainViewController.swift
//

#import <TUILiveKit/TUILiveKit-Swift.h>
```

```
- (void)buttonTapped:(UIButton *)sender {
    // RoomId is the room ID for creating a voice chat room
    NSString *roomId = @"123666";

    // Enter the voice chat room
    TUIVoiceRoomViewController *voiceRoomController = [[TUIVoiceRoomViewController


    [self.navigationController pushViewController:voiceRoomController animated:true
}
```



Voice Chat Room



Voice Chat Room

More features

Barrage

Gift

Q&A

If you encounter any problems with access and use, please refer to Q&A.

Android

Last updated：2024-05-17 11:20:40

This article will guide you through the process of quickly integrating the TUILiveKit component. By following this document, you will complete the following key steps within an hour and ultimately obtain a video or voice live streaming function with a complete UI interface.

Environment Preparations

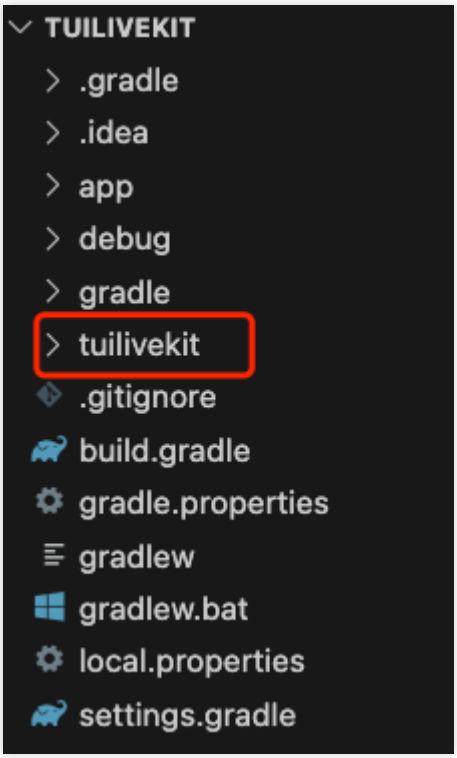
Android 5.0 (SDK API level 21) or above.
Android Studio 4.2.1 or above.
Devices with Android 5.0 or above.

Step 1. Activate the service

Before using the Audio and Video Services provided by Tencent Cloud, you need to go to the Console and activate the service for your application. For detailed steps, refer to Activate the service

Step 2: Download TUILiveKit component

Clone/download the code in Github , and then copy the tuilivekit subdirectory in the Android directory to the same level directory as the app in your current project, as shown below:



Step 3: Project configuration

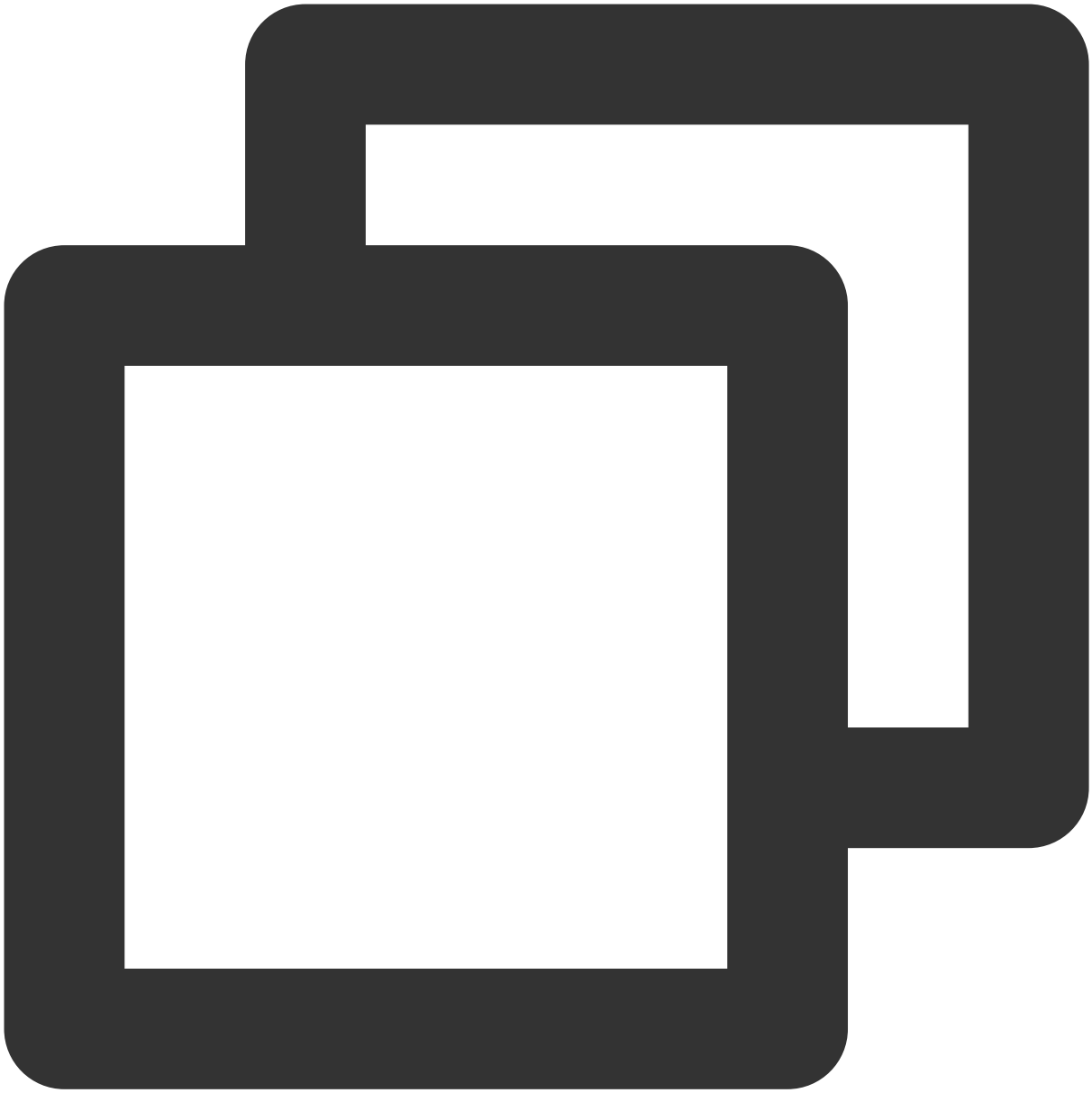
Java

1. Add `jitpack` repository dependencies to your project (Download the three-party library `SVGAPlayer` that plays gift svg animation):

Gradle 7.0 or earlier

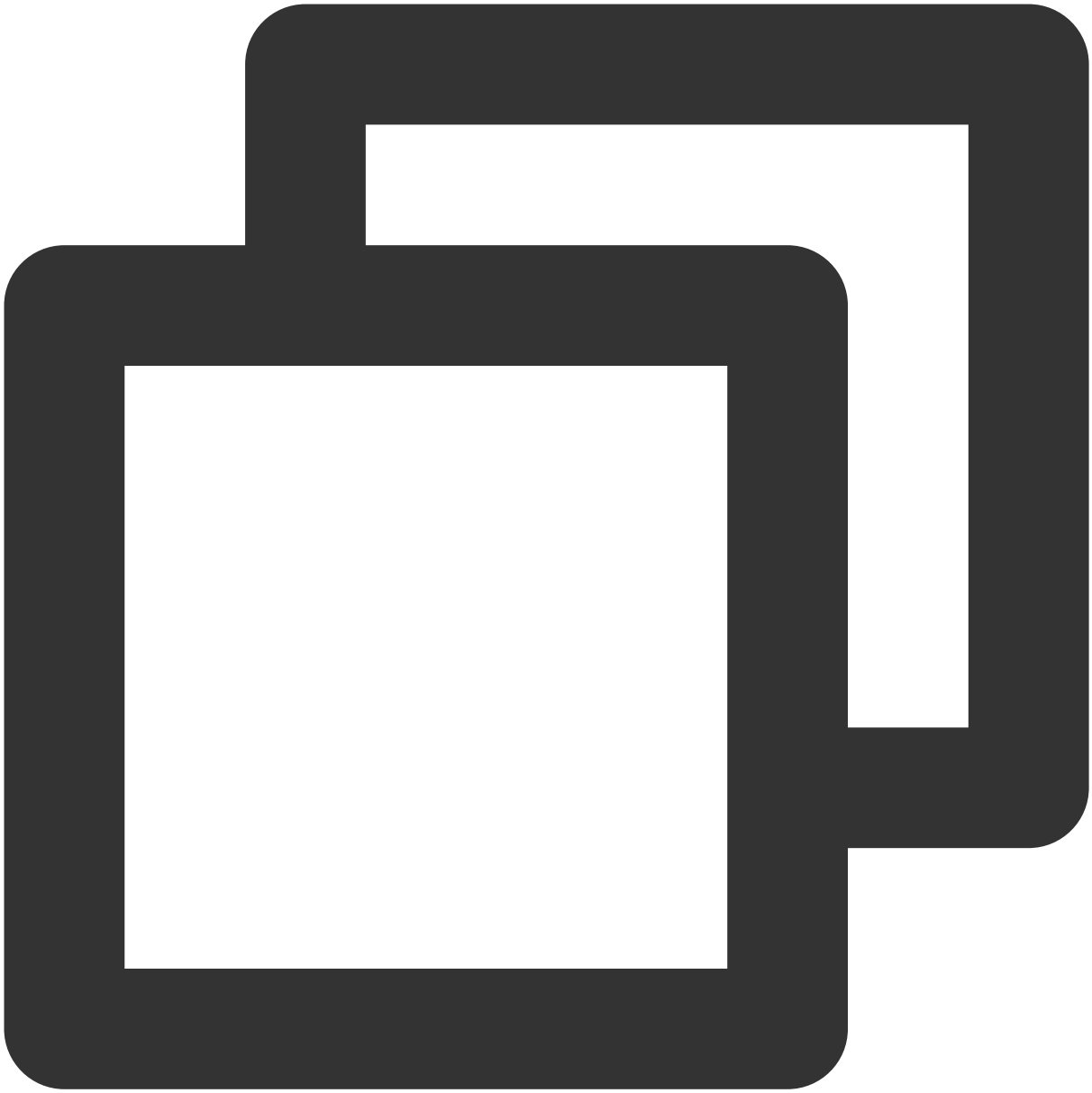
gradle 7.0 or later

Add the address of the jitpack repository to the `build.gradle` file in the root directory of the project:



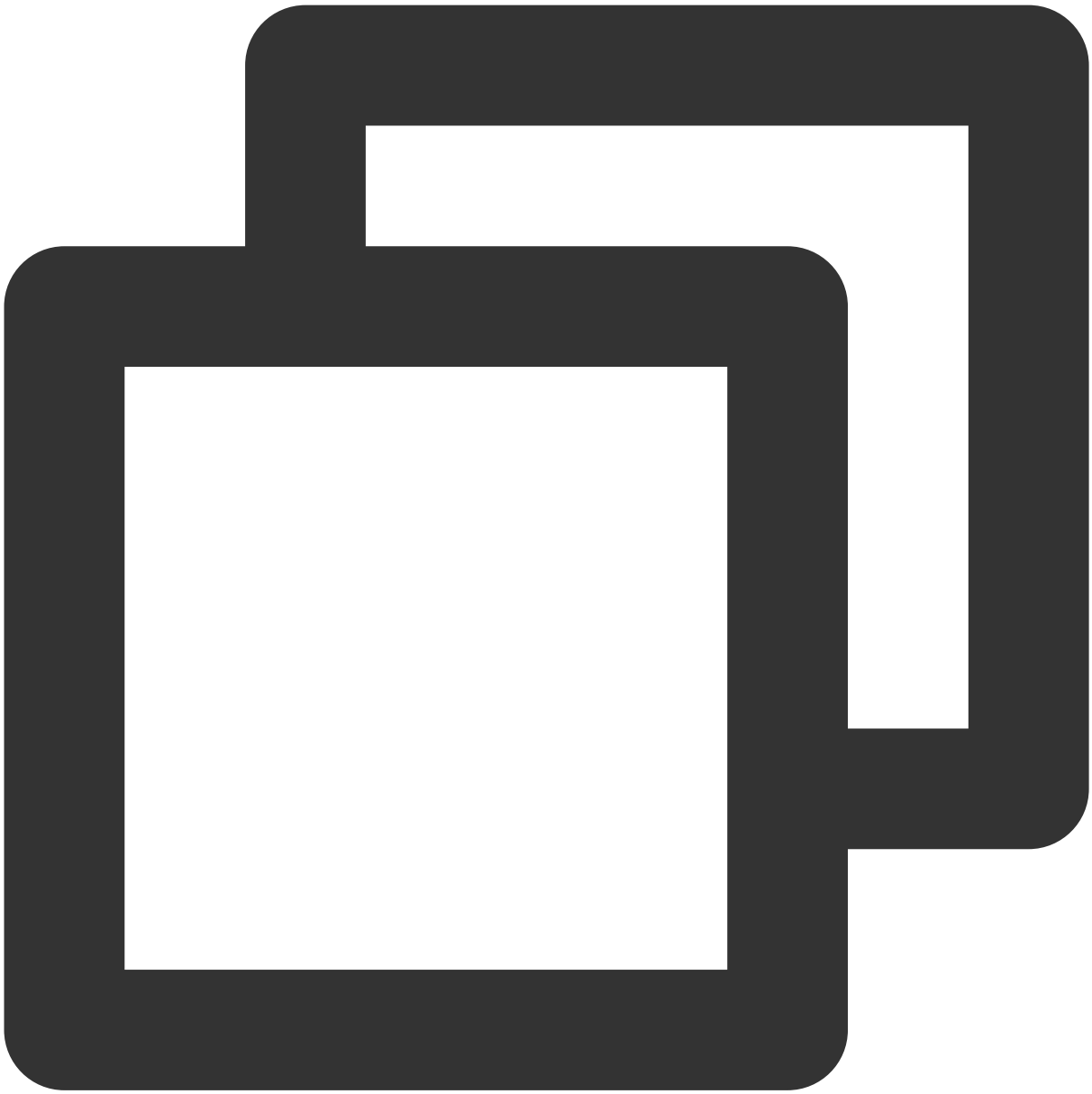
```
allprojects {
    repositories {
        google()
        mavenCentral()
        // add jitpack repository
        maven { url 'https://jitpack.io' }
    }
}
```

Add the address of the jitpack repository to the `settings.gradle` file in the root directory of the project:



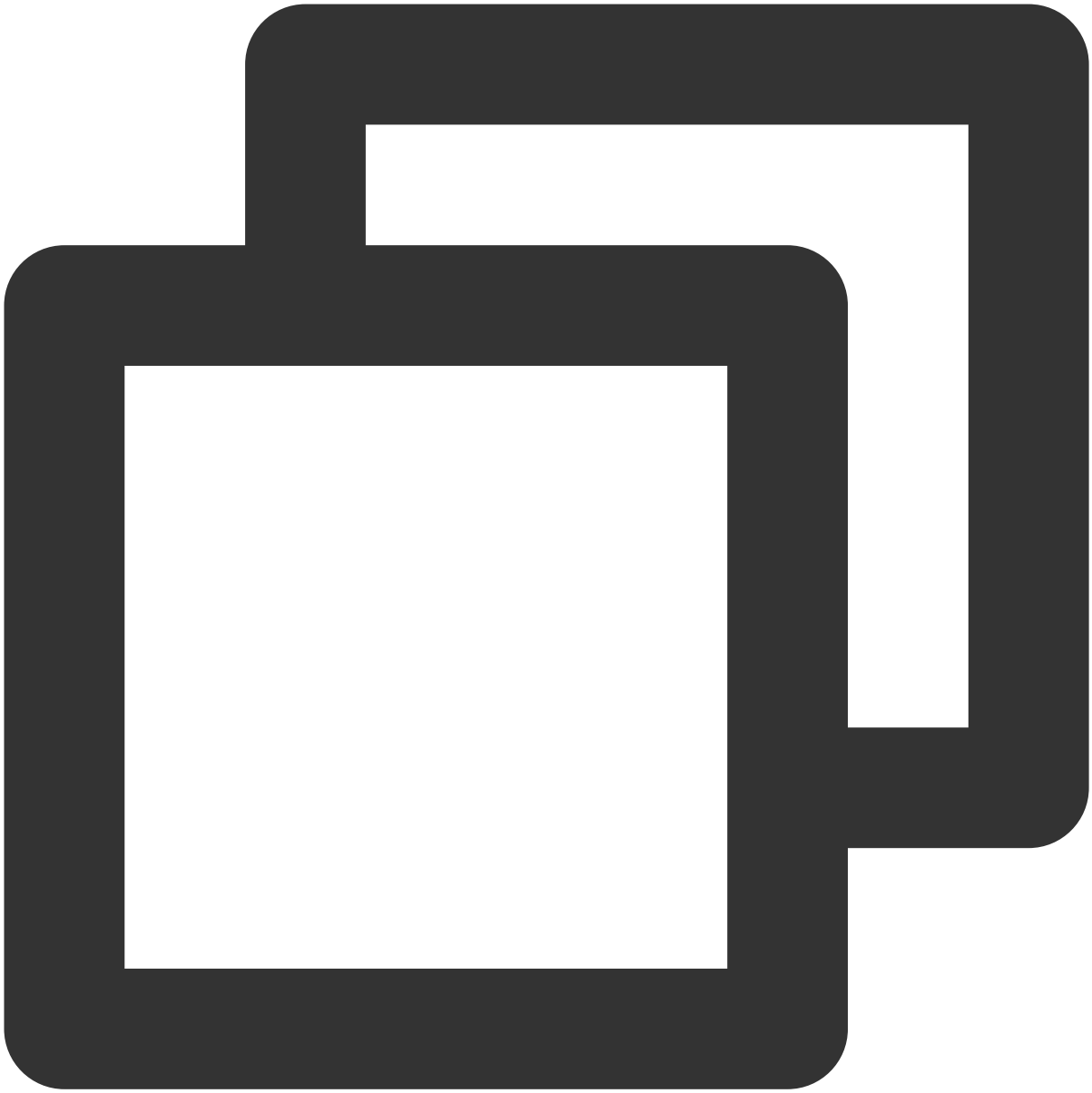
```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        // add jitpack repository
        maven { url 'https://jitpack.io' }
    }
}
```

2. Find the `settings.gradle` file in the project root directory and add the following code to it. Its function is to import the tuilivekit component downloaded in Step 2 into your current project:



```
include ':tuilivekit'
```

3. Find the `build.gradle` file in the app directory and add the following code to it. Its function is to declare the dependence of the current app on the newly added tuilivekit component:

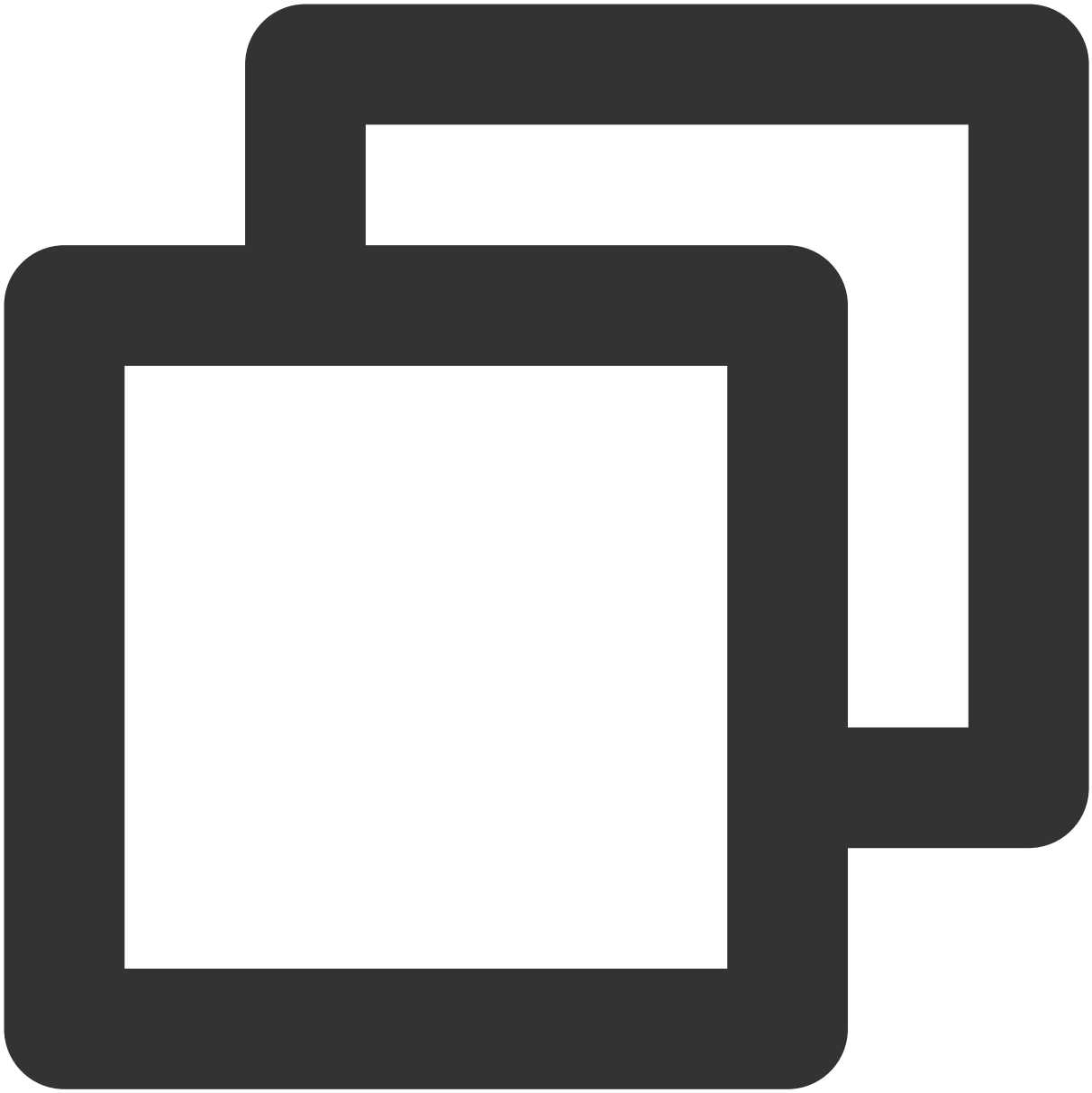


```
api project(':tuilivekit')
```

Note:

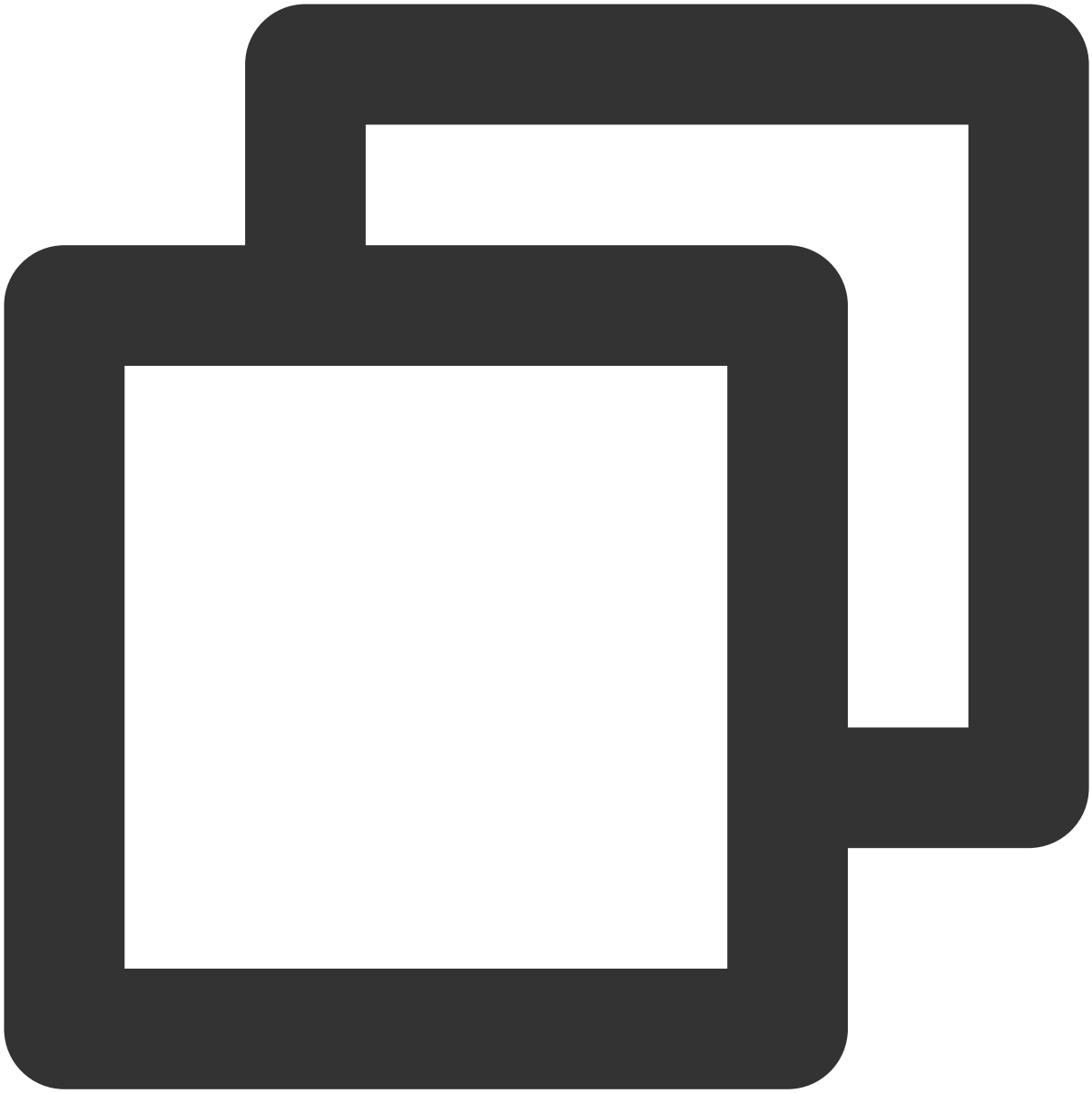
The TUILiveKit project has internal dependencies by default: `TRTC SDK`、`IM SDK`、`tuiroomengine` and the public library `tuicore`, and does not require separate configuration by developers. If you need to upgrade the version, just modify the `tuilivekit/build.gradle` file.

4. Since we use the reflection feature of Java inside the SDK, we need to add some classes in the SDK to the unobfuscated list, so you need to add the following code to the `proguard-rules.pro` file:



```
-keep class com.tencent.** { *; }
```

5. In `AndroidManifest.xml` , set a `Theme.AppCompat` Theme to the `android:theme` attribute of `application` :



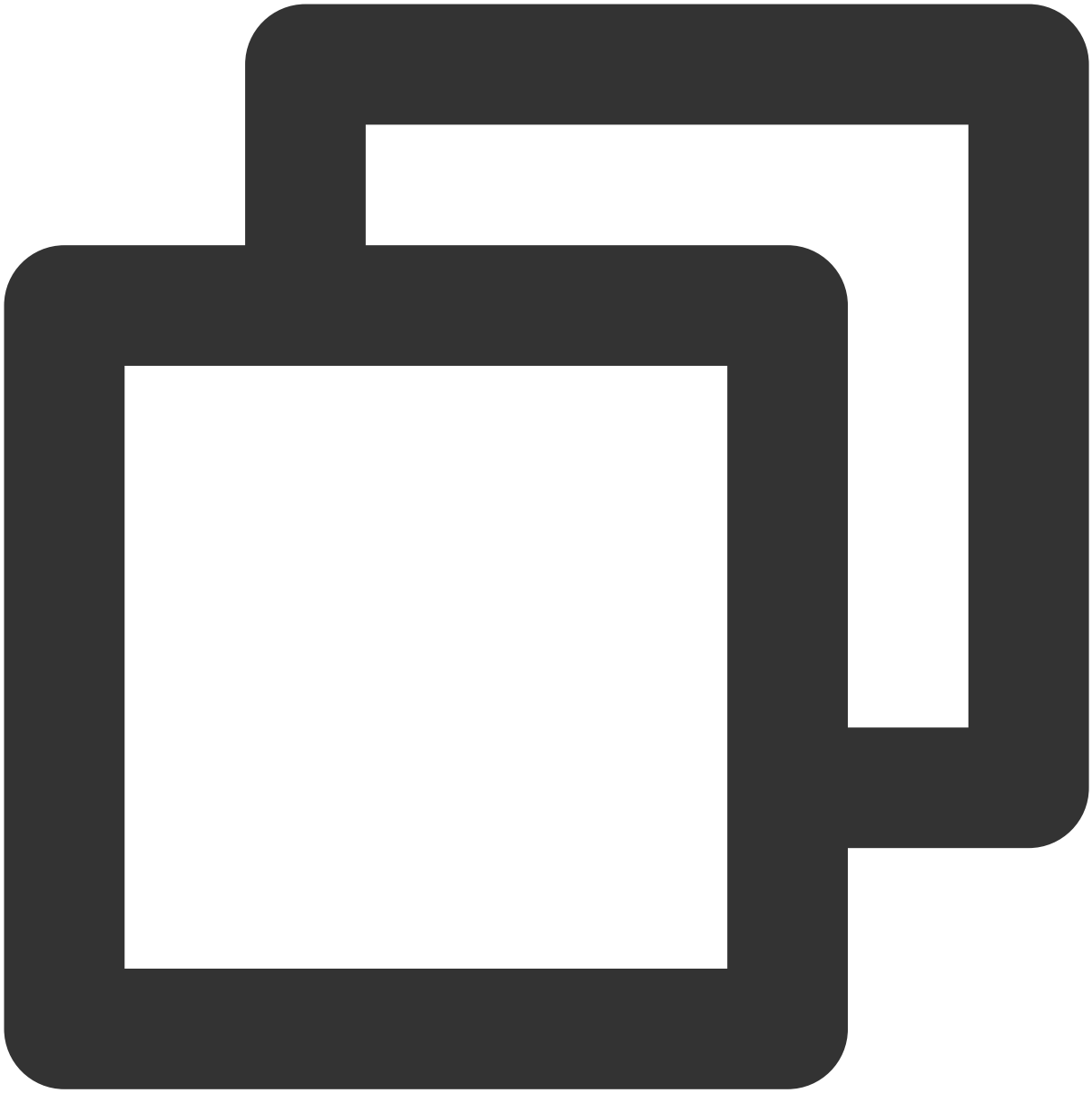
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
        ...
    </application>
</manifest>
```

Note:

TUILiveKit will internally help you dynamically apply for camera, microphone, read storage permissions, etc. If you need to delete it due to your business problems, you can modify `tuilivekit/src/main/AndroidManifest.xml` 。
If you encounter an allowBackup related exception prompted by `AndroidManifest.xml,` please refer to allowBackup Exceptions.
If you encounter problems with `Theme.AppCompat` , please refer to Activity theme issue.

Step 4. Log in

Before invoking the functions of the TUILiveKit component, you need to perform the login of the TUI component. In your project, it is recommended to add the following login code in your business login scenario or in the first startup activity of the app, which is used to complete the login of the TUI component by calling the relevant APIs in TUICore. This step is very important, because you can use all functions of TUILiveKit only after the login is successful. Therefore, please patiently check whether the relevant parameters are correctly configured.
Java



```
import TUICore

TUILogin.login(context,
    1400000001,     // Replace it with the SDKAppID obtained in Step 1
    "denny",        // Please replace it with your UserID
    "xxxxxxxxxx",  // You can calculate a UserSig in the console and fill it in
    new TUICallback() {
    @Override
    public void onSuccess() {
        Log.i(TAG, "login success");
    }
```

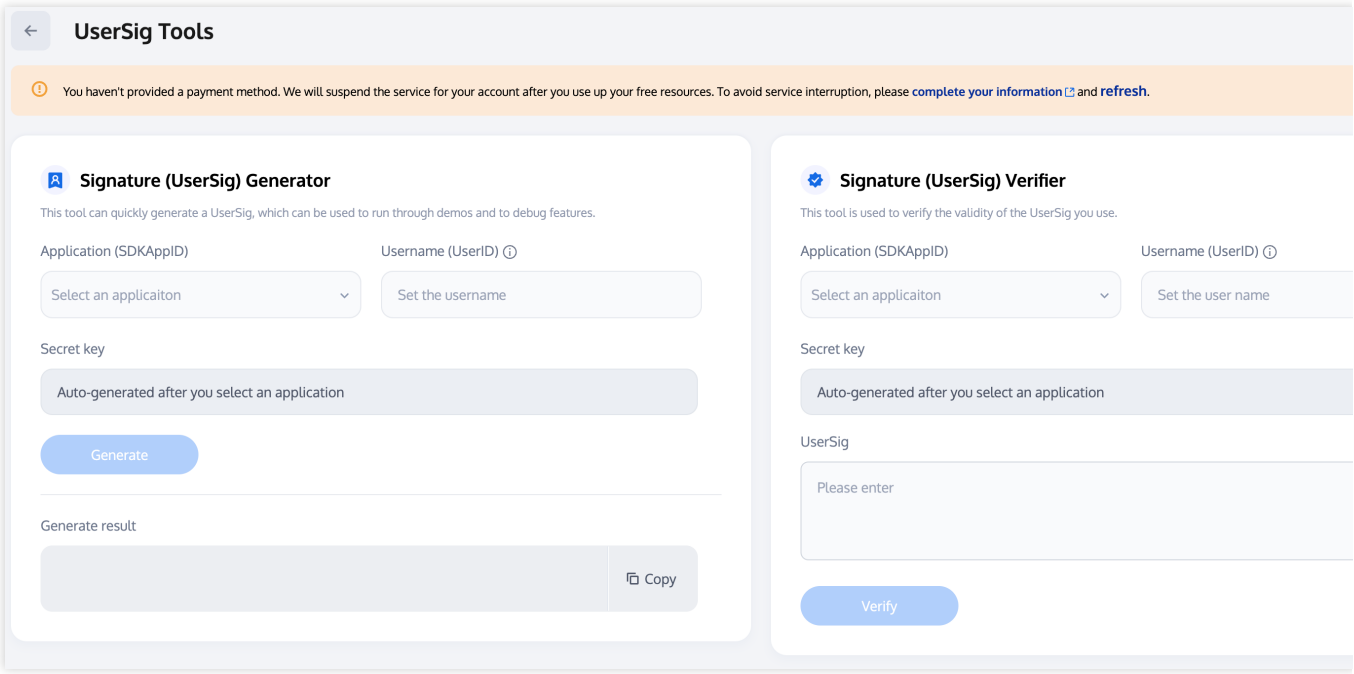
```
    @Override
    public void onError(int errorCode, String errorMessage) {
        Log.e(TAG, "login failed, errorCode: " + errorCode + " msg:" + errorMessage
    }
});
```

Parameter description: The key parameters used by the `login` function are as detailed below:

SDKAppID: Obtained in the last step in Step 1 and not detailed here.

UserID: The ID of the current user, which is a string that can contain only letters (a–z and A–Z), digits (0–9), hyphens (-), or underscores (_).

UserSig: The authentication credential used by Tencent Cloud to verify whether the current user is allowed to use the TRTC service. You can get it by using the `SDKSecretKey` to encrypt the information such as `SDKAppID` and `UserID`. You can generate a temporary `UserSig` by clicking the UserSig Generate button in the console.



For more information, see UserSig.

Note:

Many developers have contacted us with many questions regarding this step. Below are some of the frequently encountered problems:

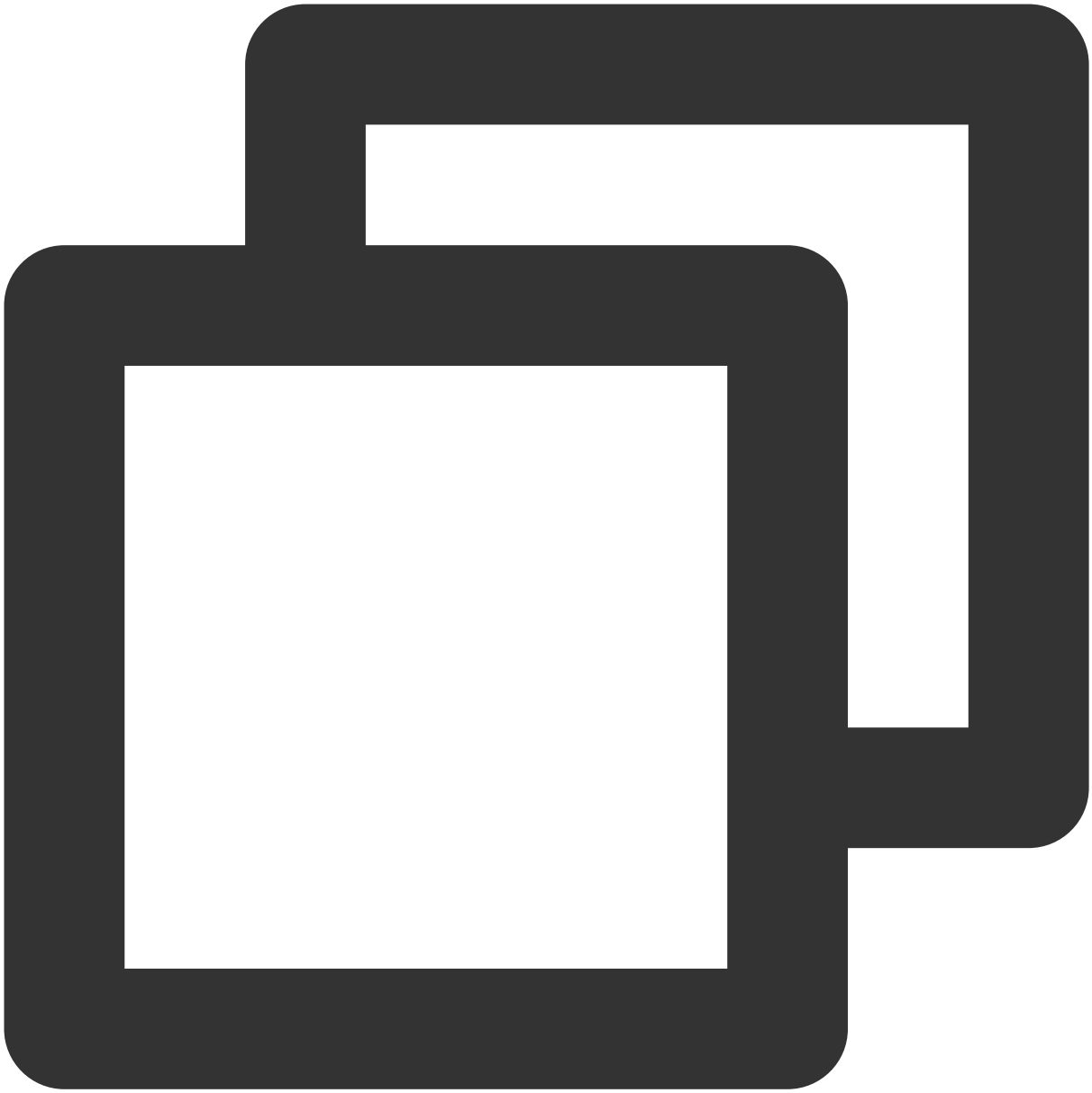
SDKAppID is invalid.

userSig is set to the value of Secretkey mistakenly. The userSig is generated by using the SecretKey for the purpose of encrypting information such as sdkAppId, userId, and the expiration time. But the value of the userSig that is required cannot be directly substituted with the value of the SecretKey.

userId is set to a simple string such as 1, 123, or 111, and your colleague may be using the same userId while working on a project simultaneously. In this case, login will fail as TRTC doesn't support login on multiple terminals with the same UserID. Therefore, we recommend you use some distinguishable userId values during debugging.
The sample code on GitHub uses the `genTestUserSig` function to calculate `UserSig` locally, so as to help you complete the current integration process more quickly. However, this scheme exposes your `SecretKey` in the application code, which makes it difficult for you to upgrade and protect your `SecretKey` subsequently. Therefore, we strongly recommend you run the `UserSig` calculation logic on the server and make the application request the `UserSig` calculated in real time every time the application uses the `TUILiveKit` component from the server.

Step 5. Enter the live preview screen

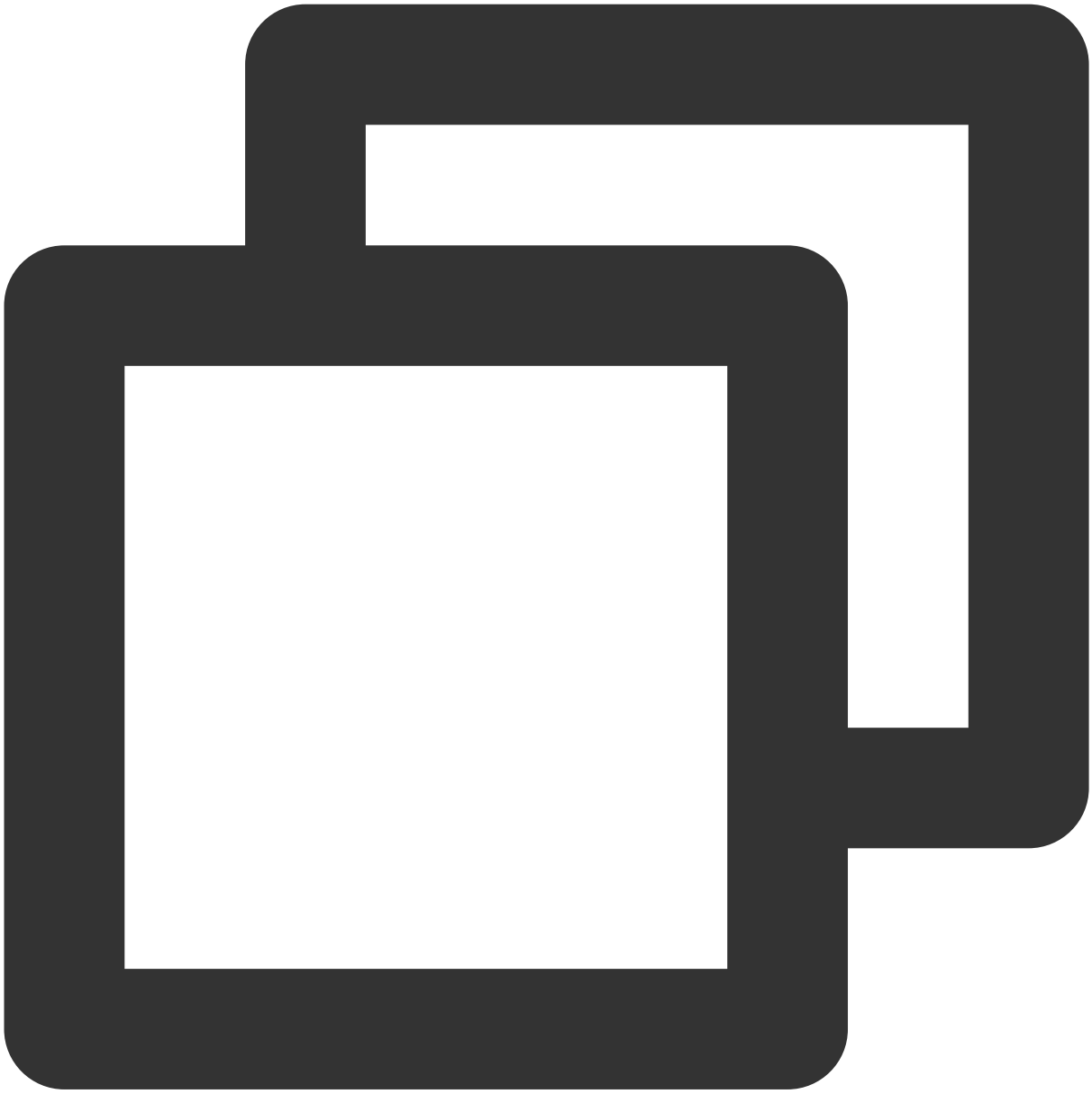
Note：
It's important to make sure you've followed Step 4 to complete the login. Only after you log in to TUILogin.login can you enter the live preview screen normally.
1. Create a new file named app_activity_anchor.xml (Default path: app/src/main/res/layout/app_activity_anchor.xml).



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/fl_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

2. Create a new file named `AnchorActivity.java` and register in the `AndroidManifest.xml`. By loading TUILiveKit TUIVoiceRoomFragment page, you can pull up preview screen.

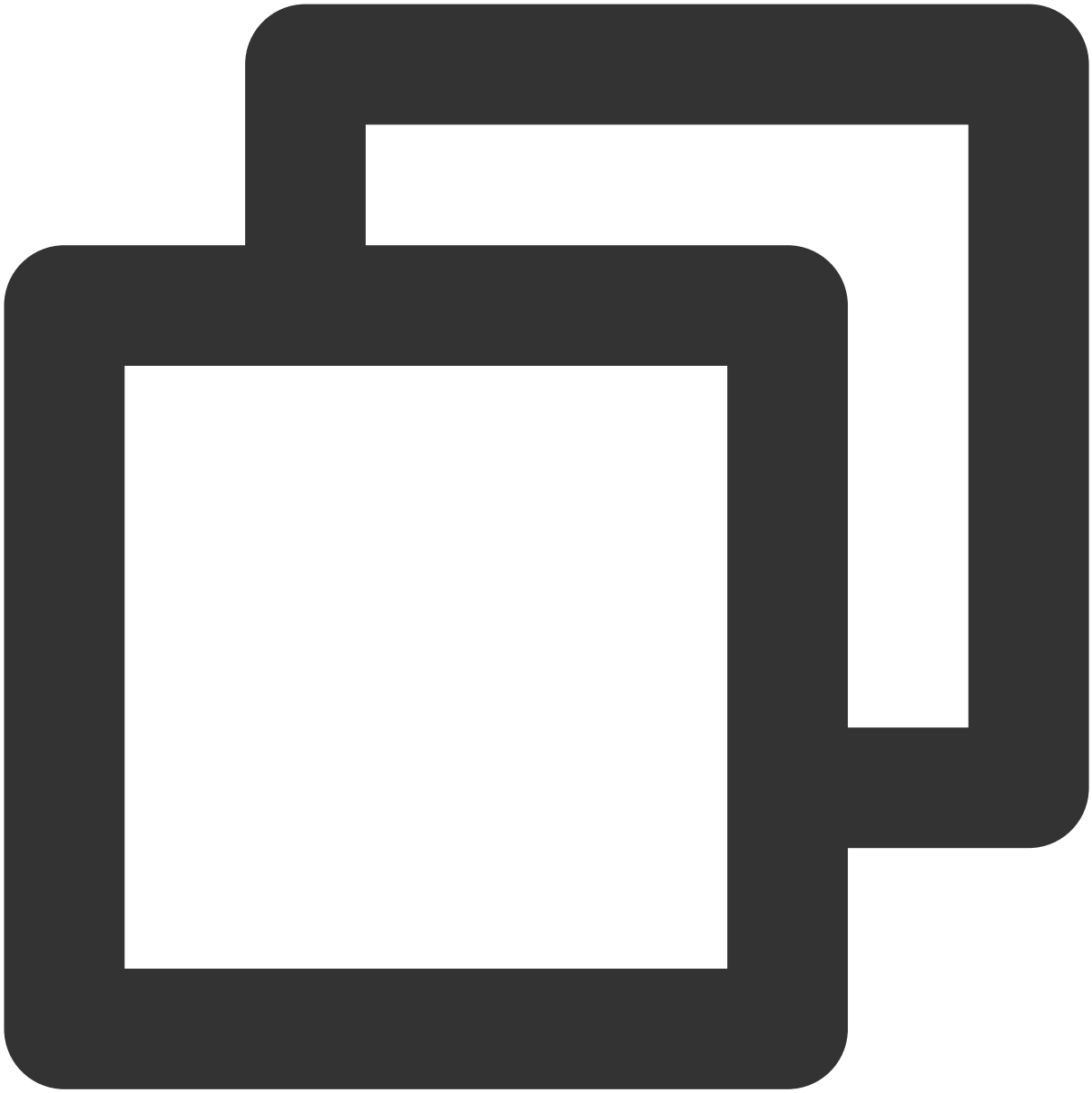
java



```
public class AnchorActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.app_activity_anchor);

        //RoomId can be customized
```

```
        String roomId = "123666";

        LiveDefine.RoomParams params = new LiveDefine.RoomParams();
        //The default value is the maximum number of seat supported by the package
        params.maxSeatCount = 0;
        params.seatMode = TUIRoomDefine.SeatMode.APPLY_TO_TAKE;

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction(
        TUIVoiceRoomFragment fragment = new TUIVoiceRoomFragment(roomId,
        LiveDefine.RoomBehavior.PREPARE_CREATE, params);
        fragmentTransaction.add(R.id.fl_container, fragment);
        fragmentTransaction.commit();
    }
}
```

Register `AnchorActivity` in `AndroidManifest.xml` of the app Project (please use the actual package name of your `AnchorActivity`):



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application>
        ...
        <!-- Example: To register AnchorActivity, please use your actual package na
        <activity android:name="com.trtc.uikit.livekit.example.main.AnchorActivity"
                  android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"/>
        ...
    </application>
</manifest>
```

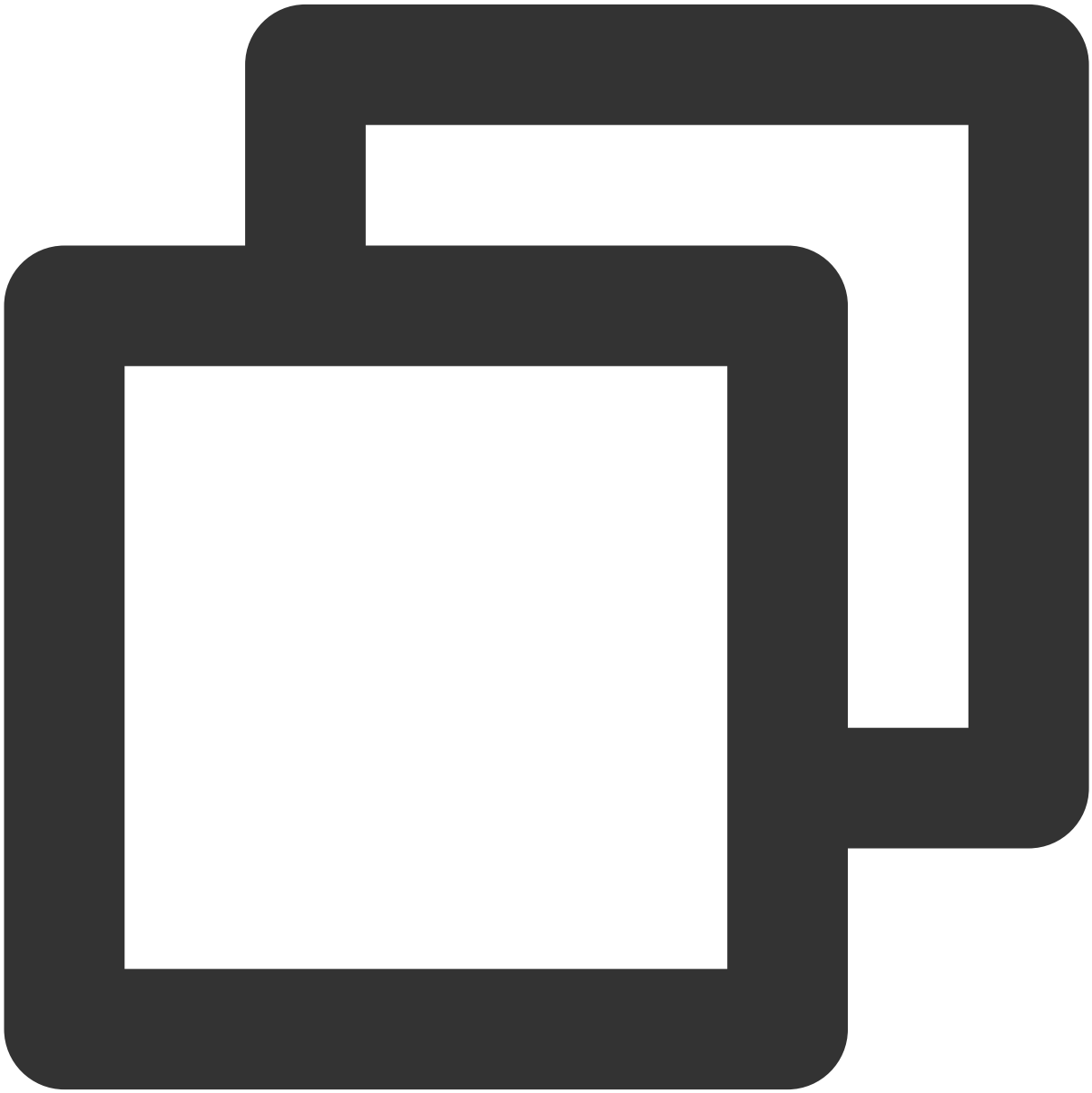
Note：

Since `AnchorActivity` inherited from `AppCompatActivity`, `AnchorActivity` was given a `Theme.AppCompat` theme. You can modify it to your own `Theme.AppCompat` theme.

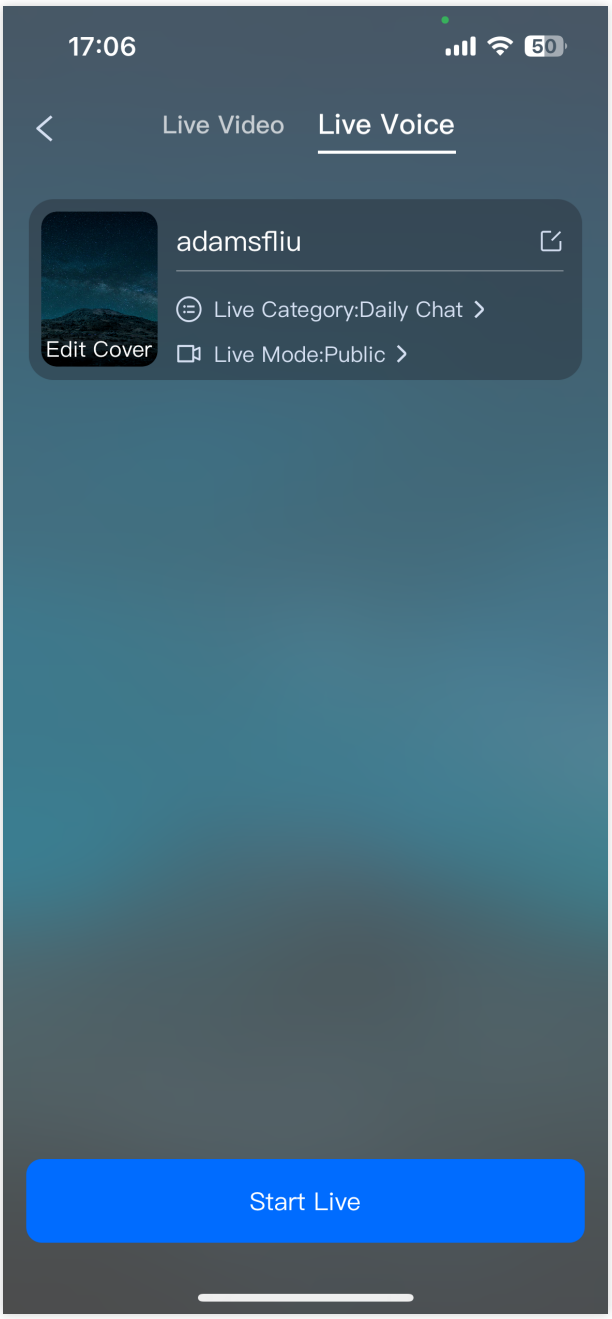
If you encounter problems with Theme.AppCompat, please refer to Activity theme issue.

3. Where you need to start live streaming (depending on your business, it can be executed in a click event in MainActivity by default), perform the following operations to pull up the host start page:

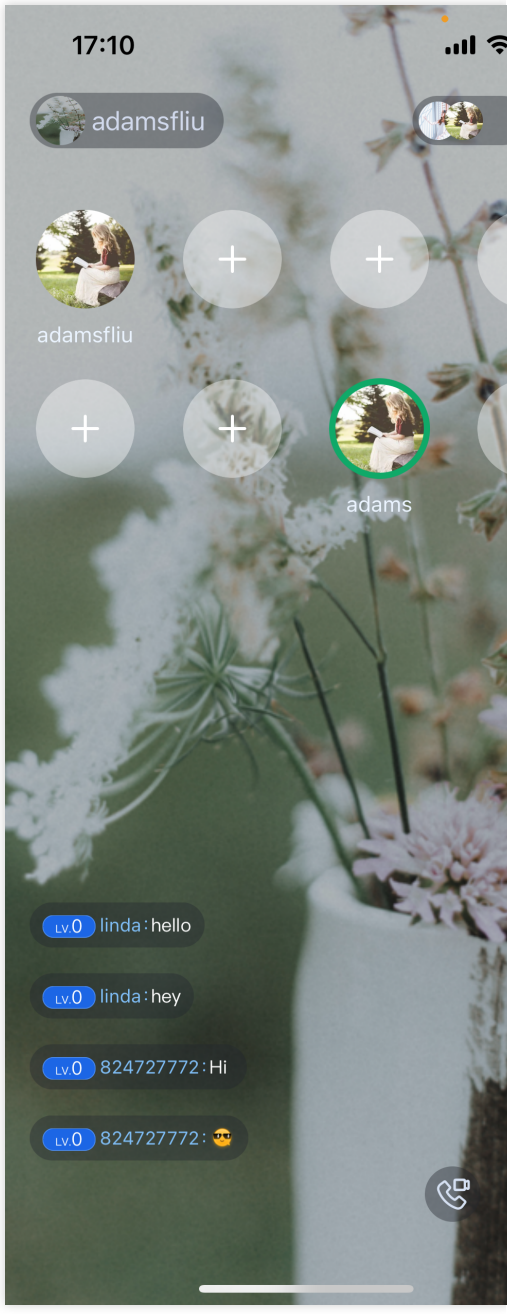
Java



```
Intent intent = new Intent(context, AnchorActivity.class);
startActivity(intent);
```



Voice chat room preview screen



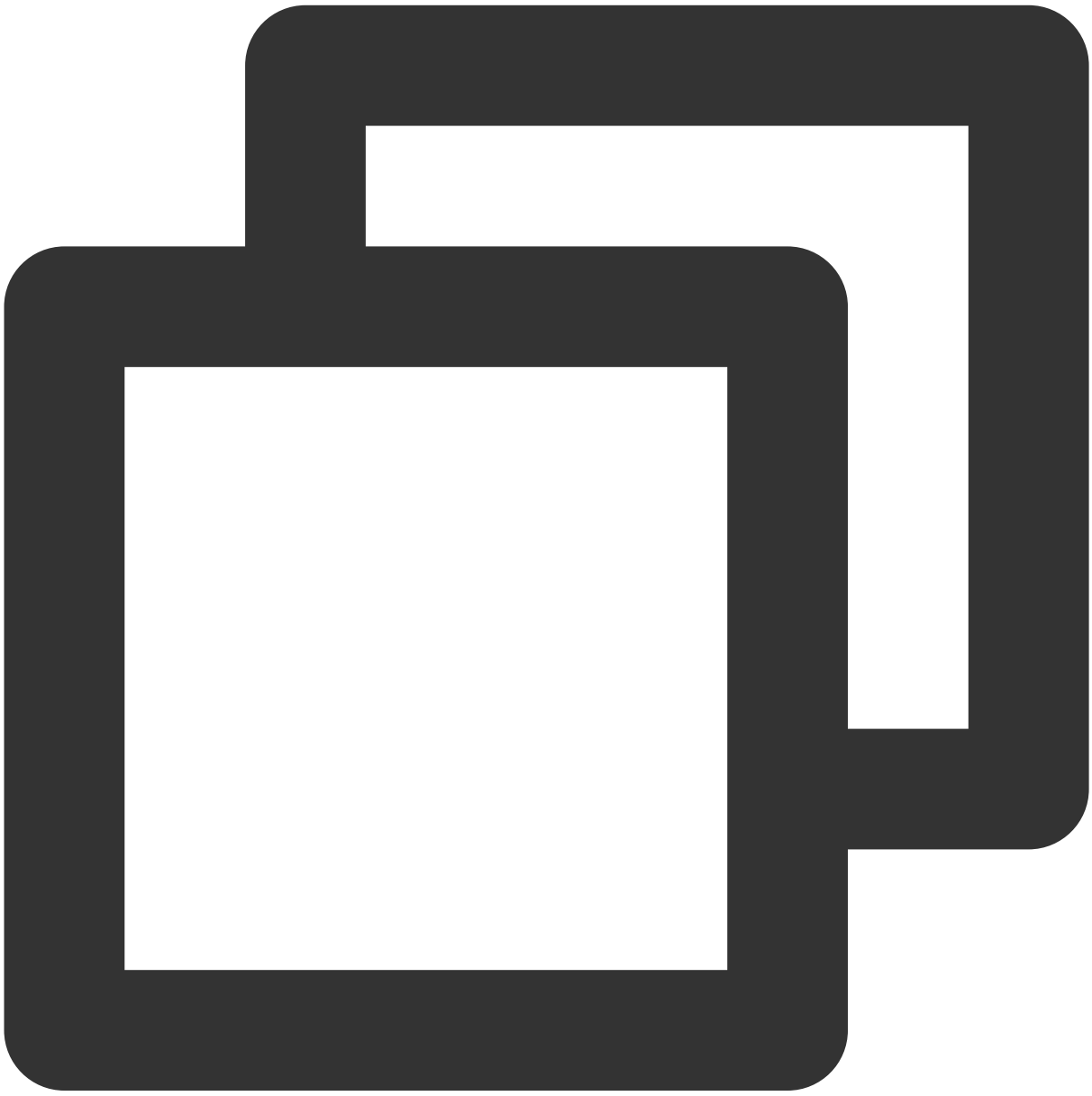
Voice chat room in-room screen

Step 6. The audience enters the studio

Note：

It's important to make sure you've followed Step 4 to complete the login. Only after you log in to TUILogin.login can you enter the live preview screen normally. Also, the audience UserID should not be the same as the anchor UserID.

1. Create a new file named app_activity_audience.xml (Default path: app/src/main/res/layout/app_activity_audience.xml).

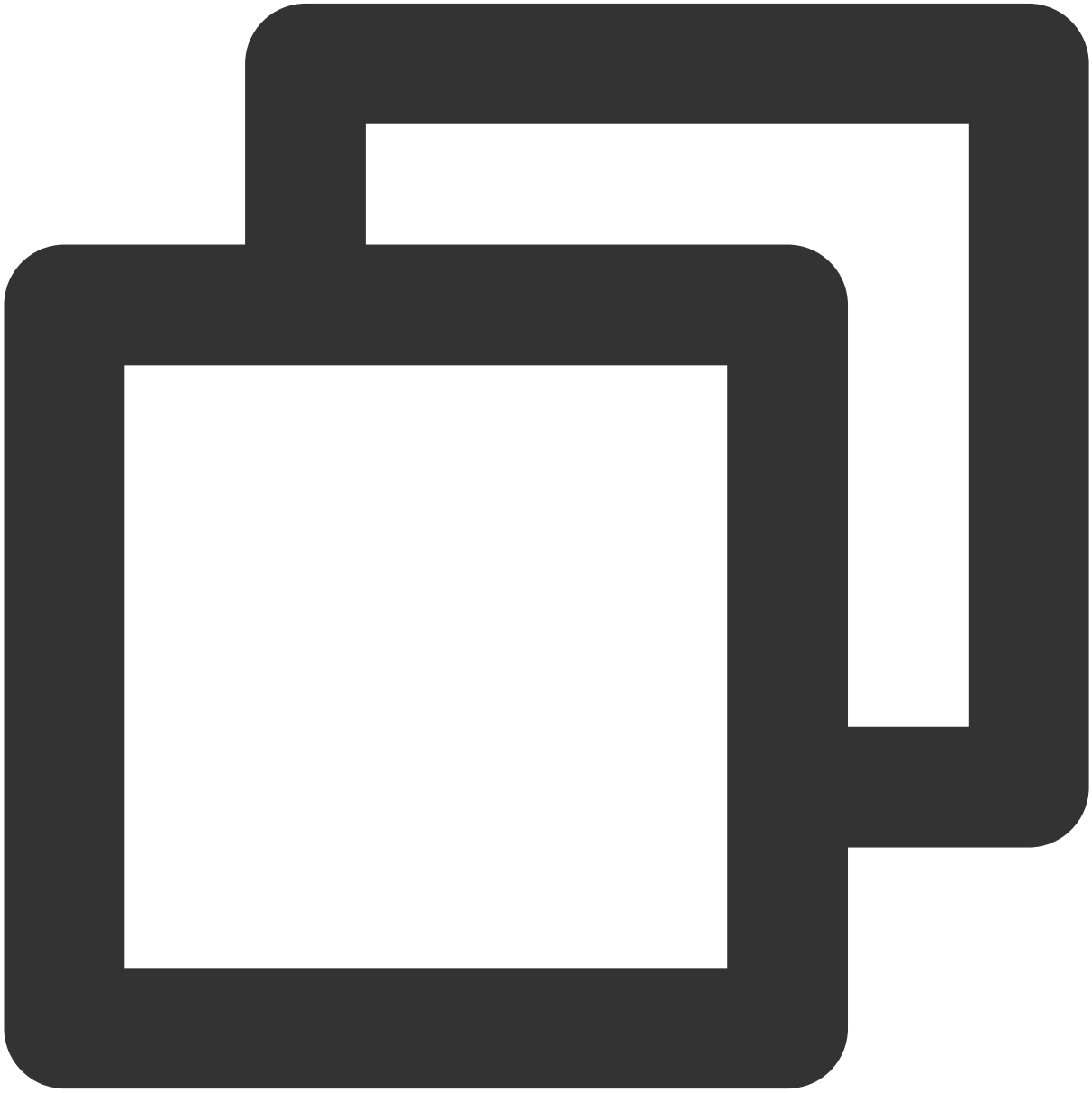


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
    <FrameLayout
        android:id="@+id/fl_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

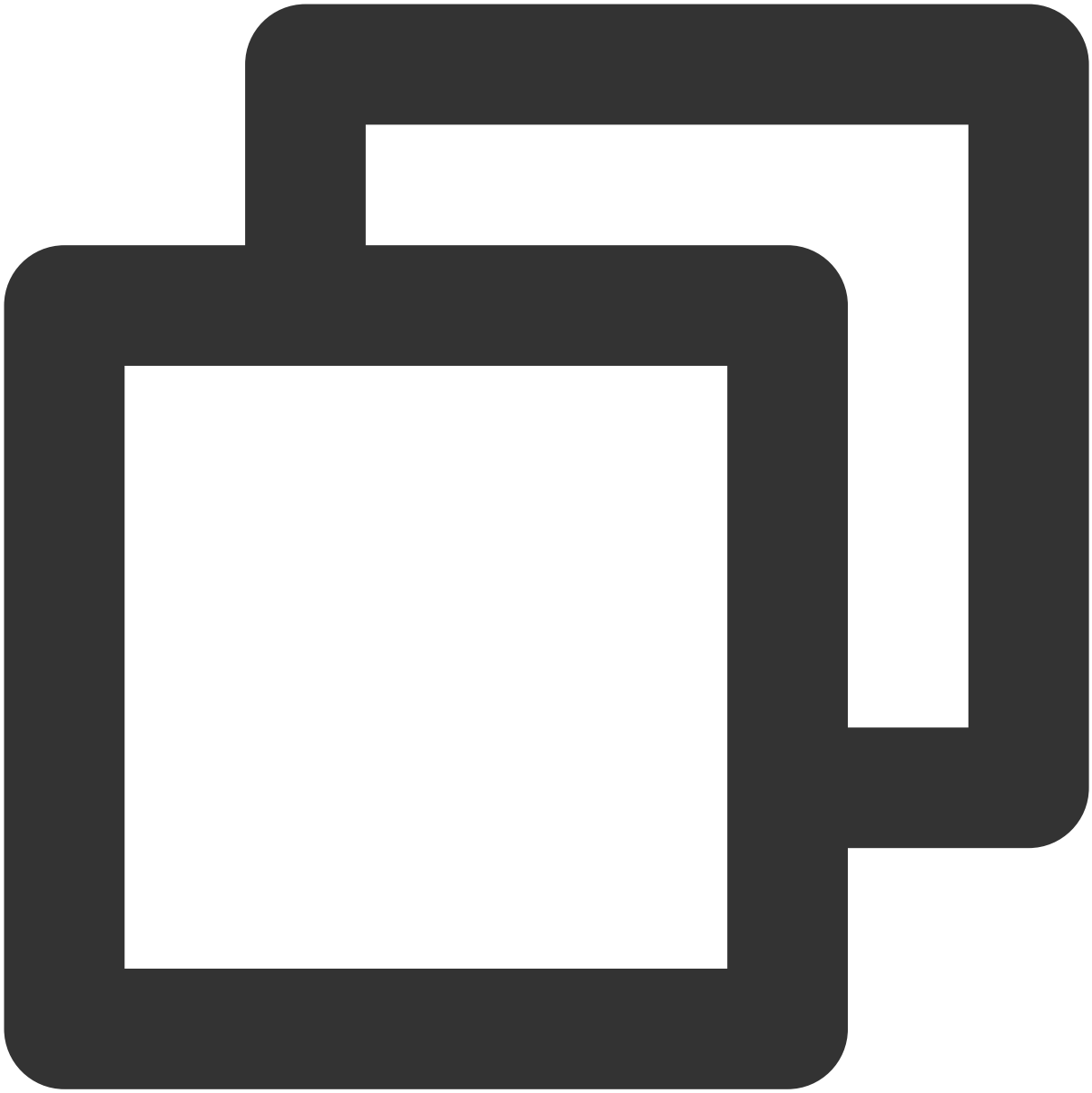
2. Create a new file named `AudienceActivity.java` and register in the `AndroidManifest.xml` . By loading TUILiveKit `TUILiveRoomAudienceFragment` page, you can enter room.

java



```
public class AudienceActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.app_activity_audience);
        //RoomId can be customized
        String roomId = "123666";

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction(
        TUIVoiceRoomFragment fragment = new TUIVoiceRoomFragment(roomId,
                LiveDefine.RoomBehavior.JOIN, null);
        fragmentTransaction.add(R.id.fl_container, fragment);
        fragmentTransaction.commit();
    }
}
```

Register `AudienceActivity` in `AndroidManifest.xml` of the app Project (please use the actual package name of your `AudienceActivity`):



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application>
        ...
        <!-- Example: To register AudienceActivity, please use your actual package
        <activity android:name="com.trtc.uikit.livekit.example.main.AudienceActivit
                  android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"/>
        ...
    </application>
</manifest>
```

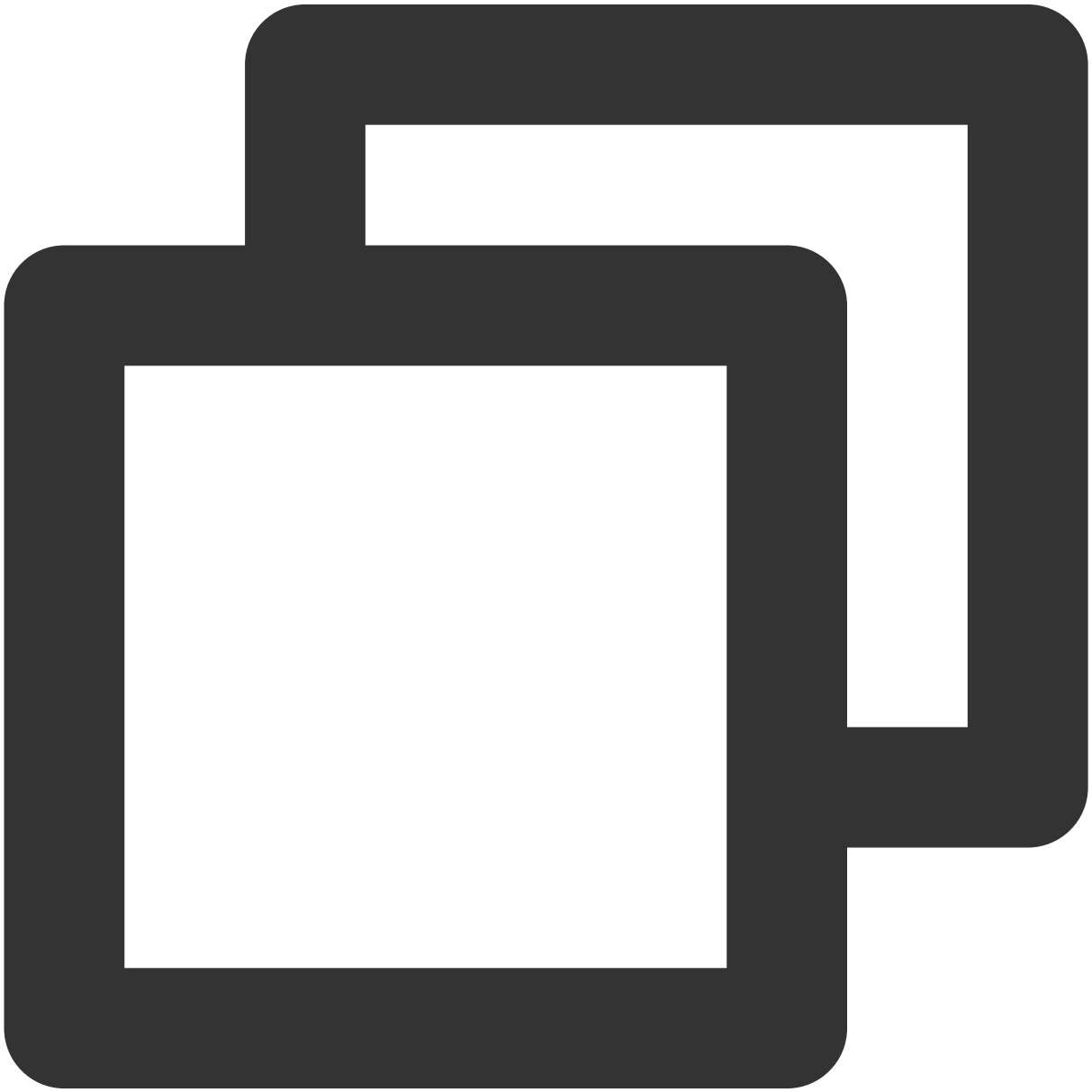
Note：

Since `AnchorActivity` inherited from `AudiencetActivity` , `AudiencetActivity` was given a `Theme.AppCompat` theme. You can modify it to your own `Theme.AppCompat` theme.

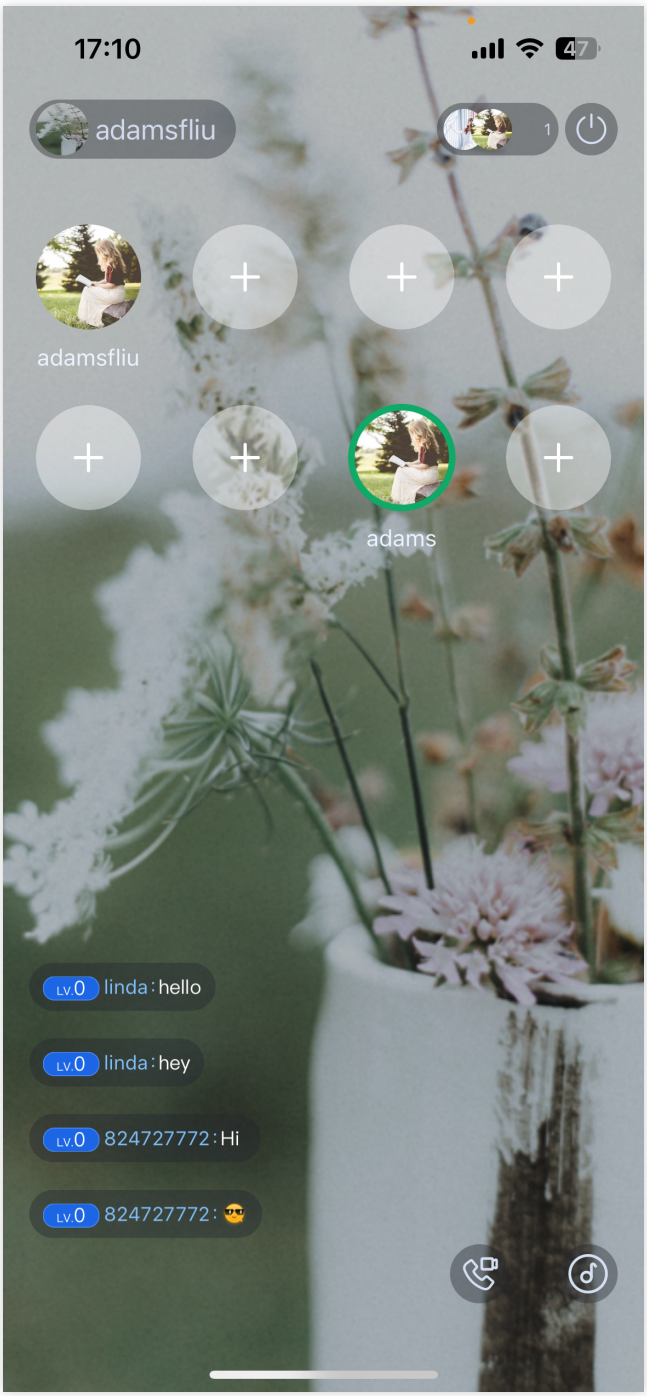
If you encounter problems with Theme.AppCompat, please refer to Activity theme issue.

3. Where you need the audience to enter the room (depending on your business, it can be executed in a click event in MainActivity by default), do the following to pull up the audience entry page:

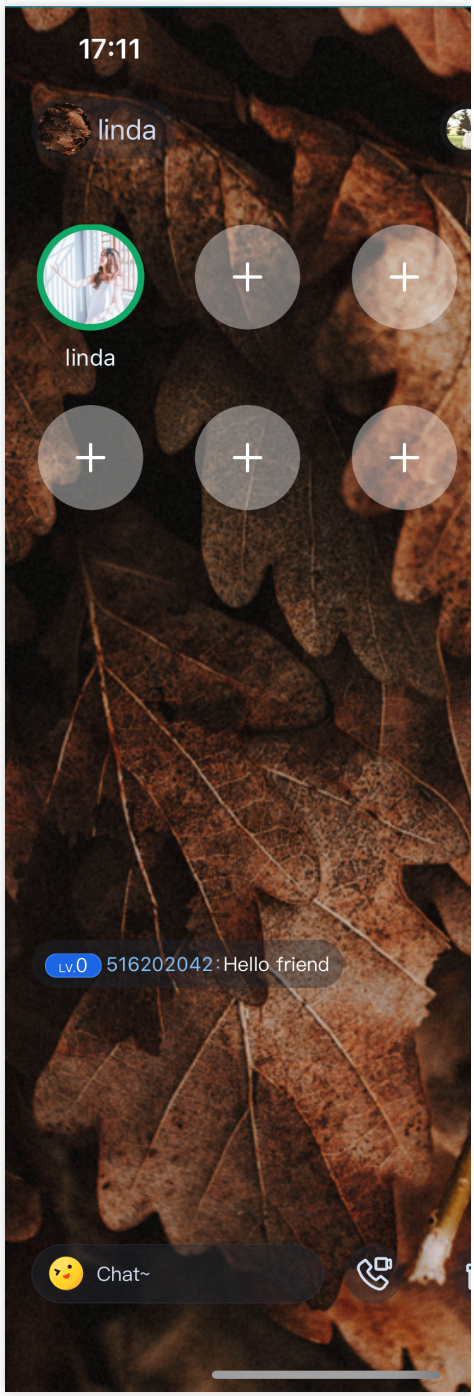
Java



```
Intent intent = new Intent(context, AudienceActivity.class);
startActivity(intent);
```



Voice Chat Room



Voice Chat Room

More features

Barrage
Gift

Q&A

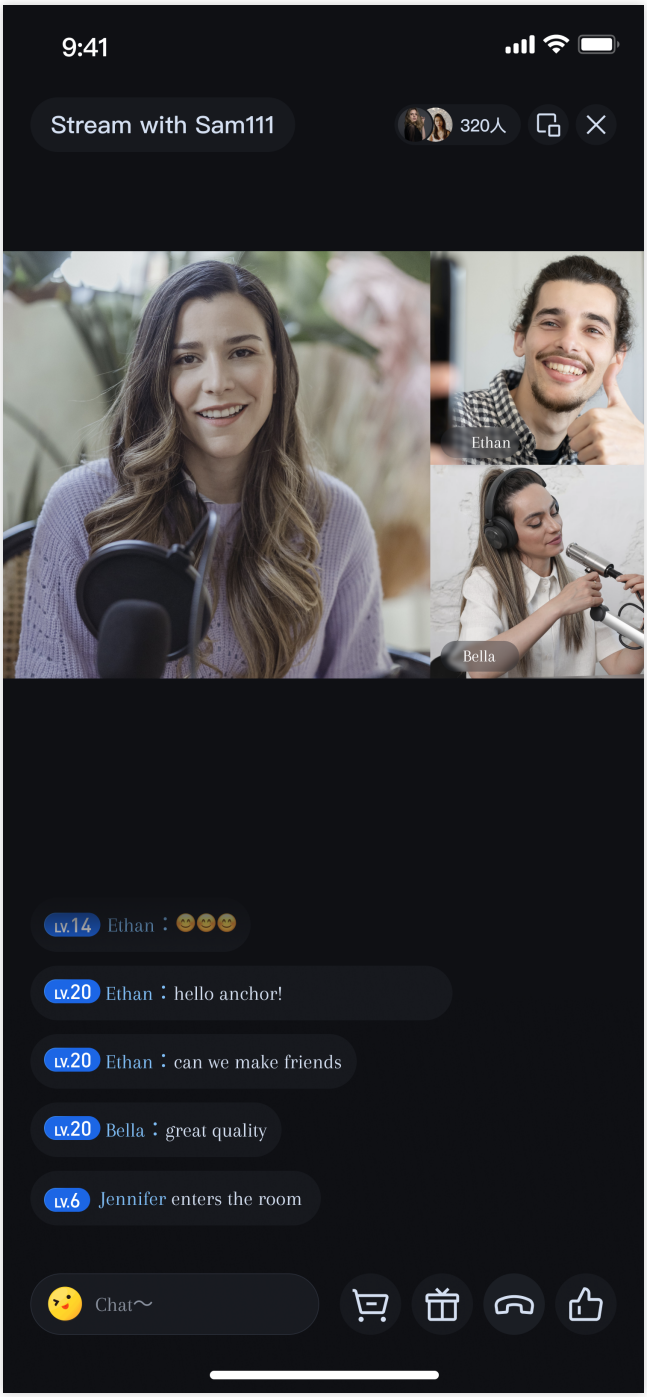
If you encounter problems with access and use, see Q&A。

Interactive Bullet Comments (TUILiveKit)
iOS

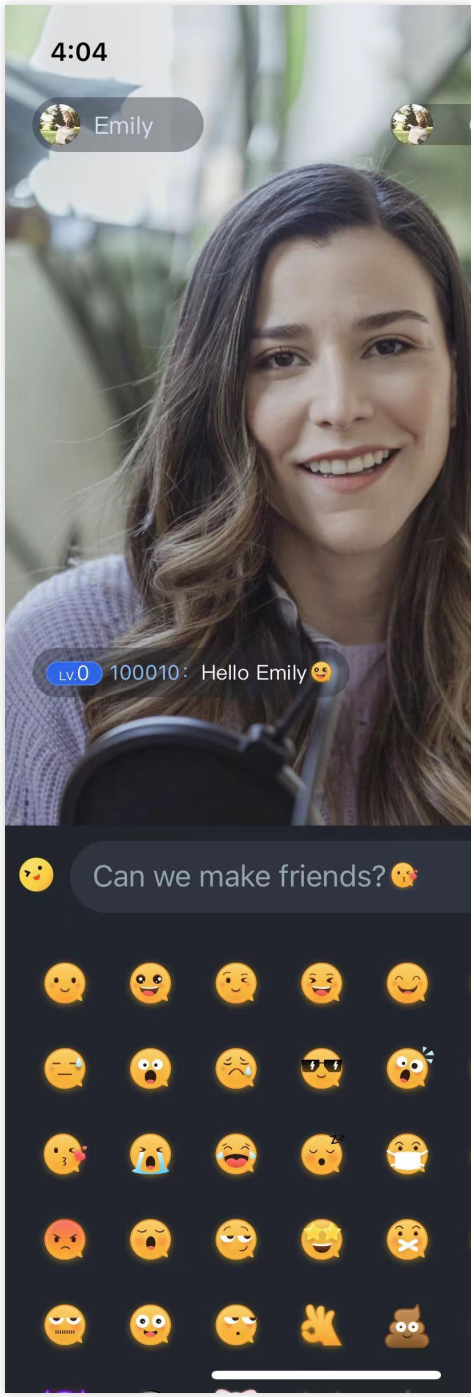
Last updated：2024-08-09 22:25:01

Overview

Live Chat feature supports the following functions: sending barrage messages, inserting custom messages, and custom message styles. Live Chat messages support emoji input, adding fun to the messages and making the interaction more enjoyable.



Display barrage



Send barrage

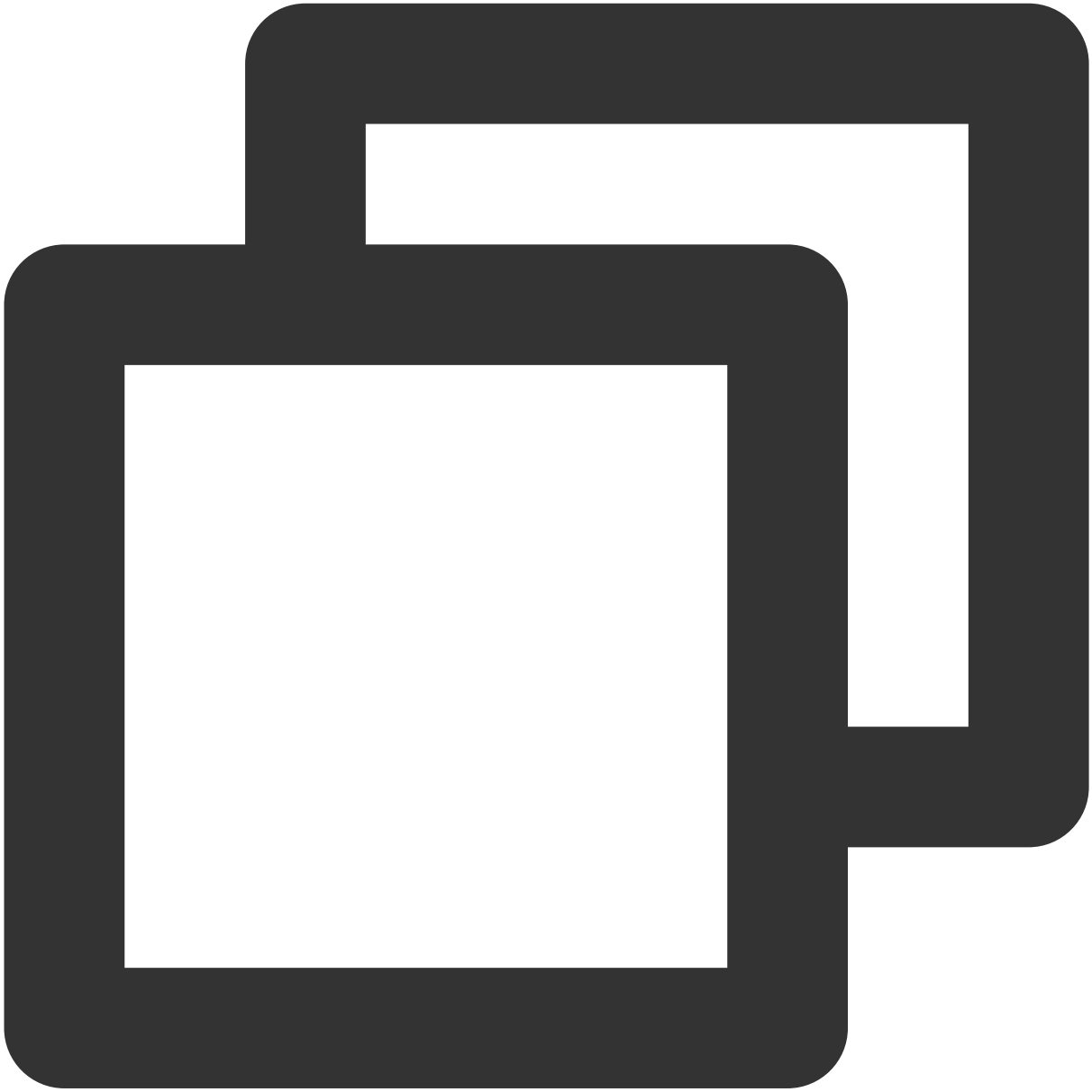
Note：

Support switching between **system keyboard** and **emoji keyboard**.

Integration

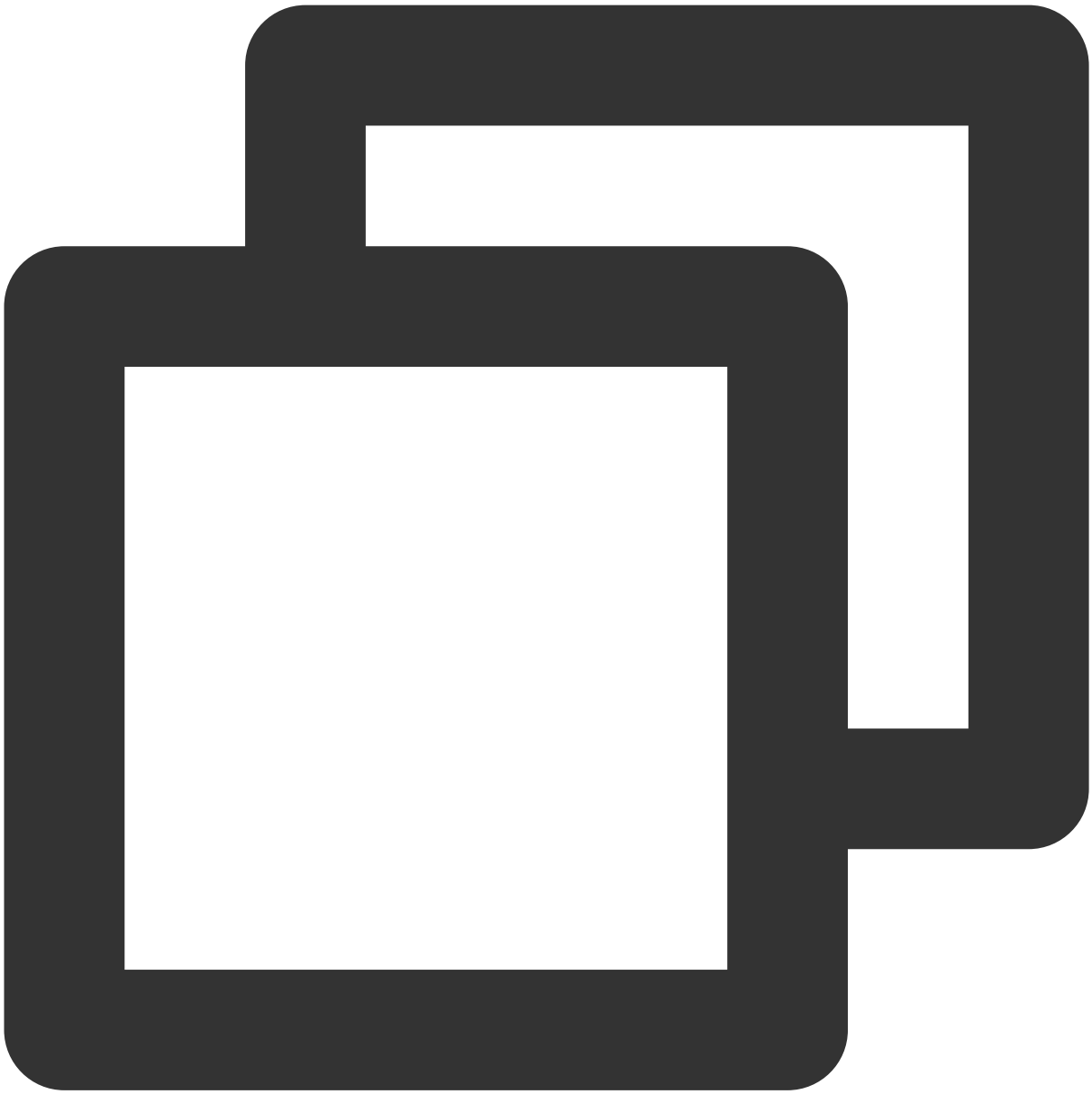
The barrage component mainly provides two Objects：

`TUIBarrageButton` : Clicking it can bring up the input interface.
`TUIBarrageDisplayView` : Used for displaying barrage messages.
In scenarios where barrage messages need to be sent, create a `TUIBarrageButton` , which can bring up the input interface when clicked:



```
let barrageButton: TUIBarrageButton = TUIBarrageButton(roomId: xxx)
view.addSubview(barrageButton)
// layout barrageButton
```

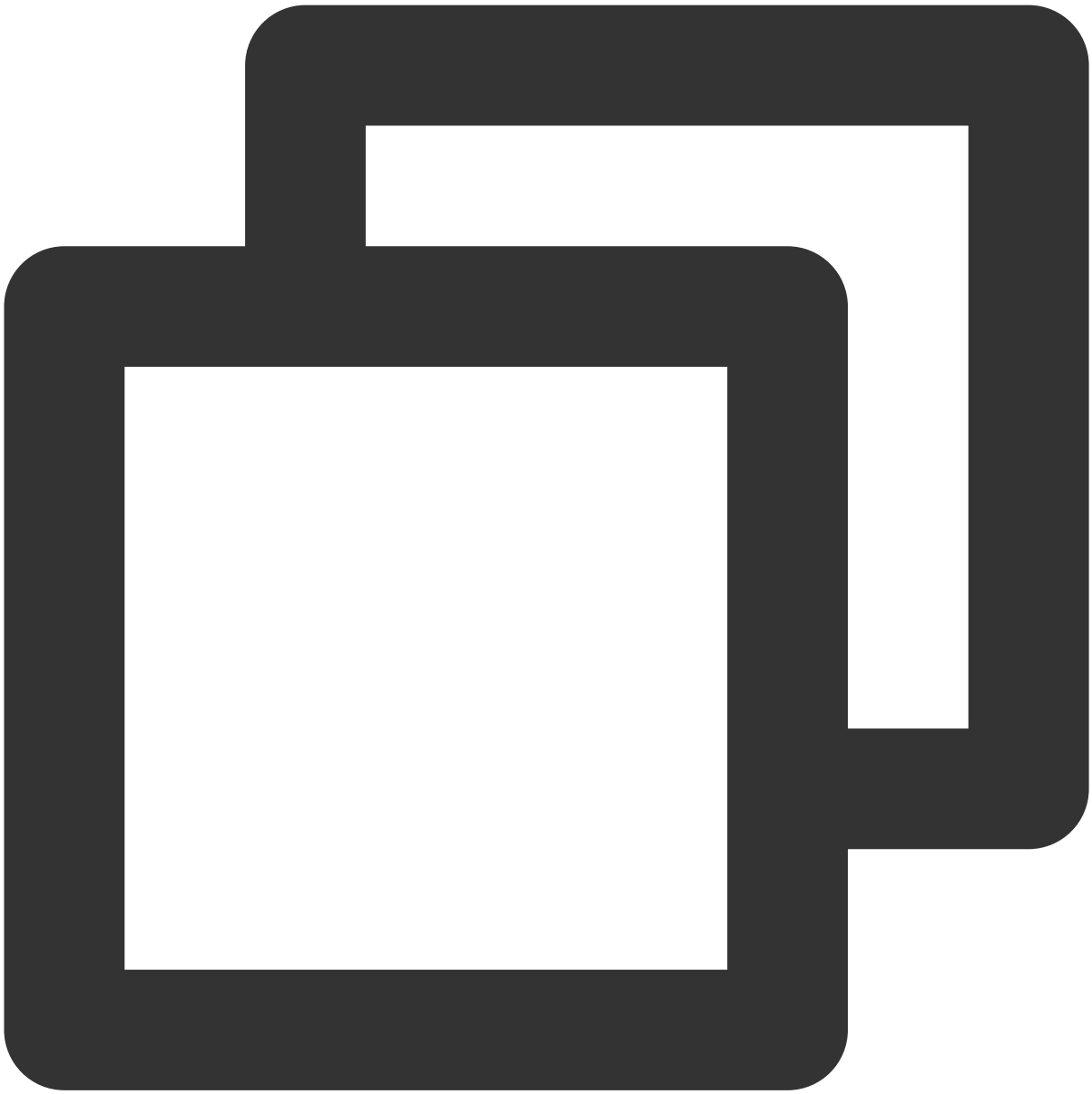
In scenarios where barrage messages need to be displayed, use `TUIBarrageDisplayView` to show the barrage messages:



```
let barrageDisplayView: TUIBarrageDisplayView = TUIBarrageDisplayView(roomId: xxx)
view.addSubview(barrageDisplayView)
// layout barrageDisplayView
```

Customize message style

Implement the `createCustomCell` delegate function in the `TUIBarrageDisplayViewDelegate` of `TUIBarrageDisplayView`, which is used to customize the barrage message style.



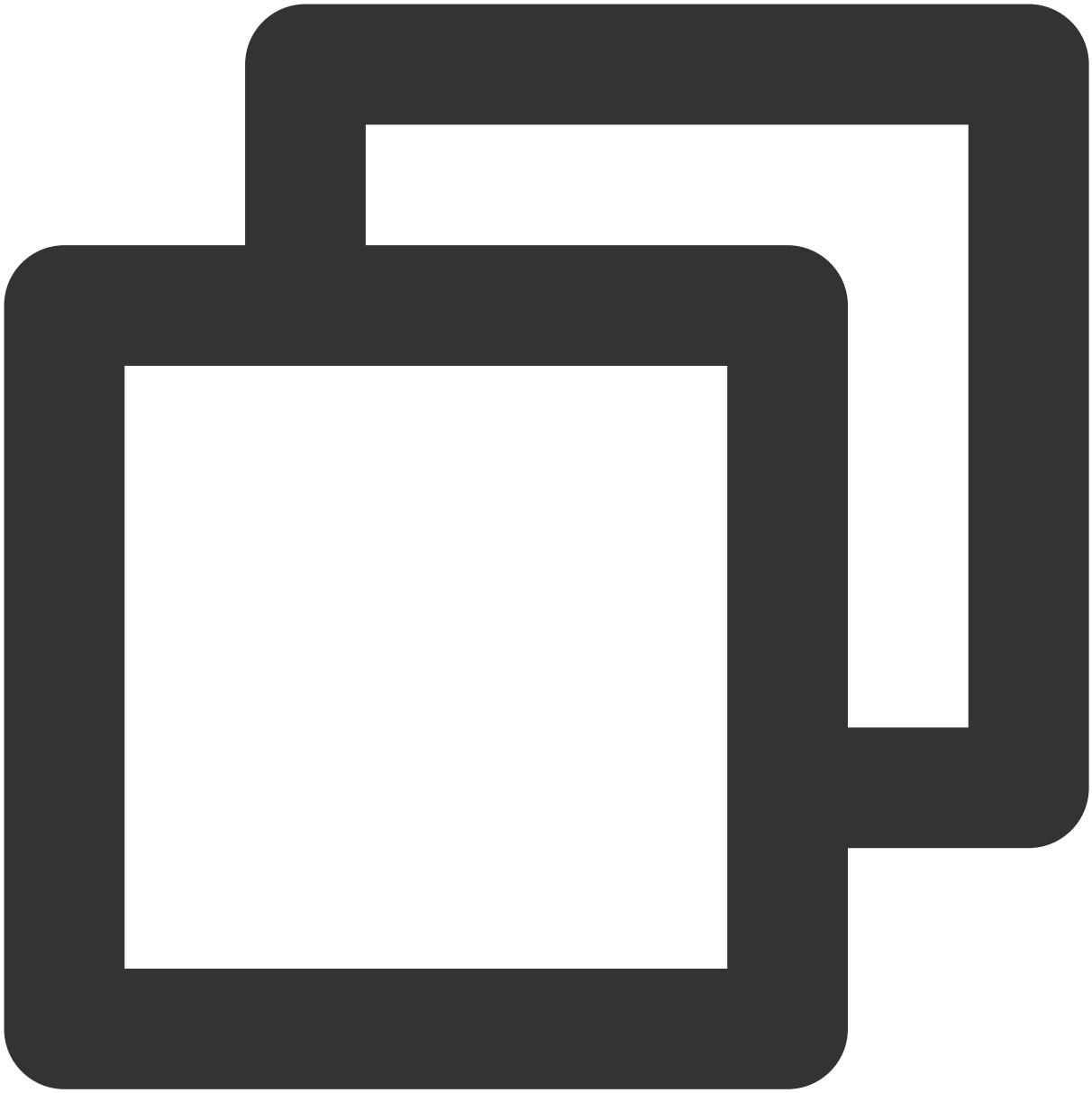
```
barrageDisplayView.delegate = self
extension UIViewController: TUIBarrageDisplayViewDelegate {
   func barrageDisplayView(_ barrageDisplayView: TUIBarrageDisplayView, createCusto
       // Return custom barrage UI here.
    }
}
```

Note：

When displaying messages, `TUIBarrageDisplayView` will first call the delegate function `barrageDisplayView:createCustomCell` to obtain the user's custom style for a specific barrage message. If it returns nil, the default barrage style of TUIBarrageDisplayView will be used.

InsertCustomMessage

The barrage display component `TUIBarrageDisplayView` provides the external interface method `insertBarrages` for inserting custom messages (in batches). Custom messages are usually used in combination with custom styles to achieve different display effects.



```
// Example: Insert a gift message in the barrage area.
let barrage = TUIBarrage()
barrage.content = "gift"
barrage.user.userId = sender.userId
barrage.user.userName = sender.userName
barrage.user.avatarUrl = sender.avatarUrl
barrage.user.level = sender.level
barrage.extInfo["xxx"] = "xxx"
barrageDisplayView.insertBarrages(barrage);
```

Note：

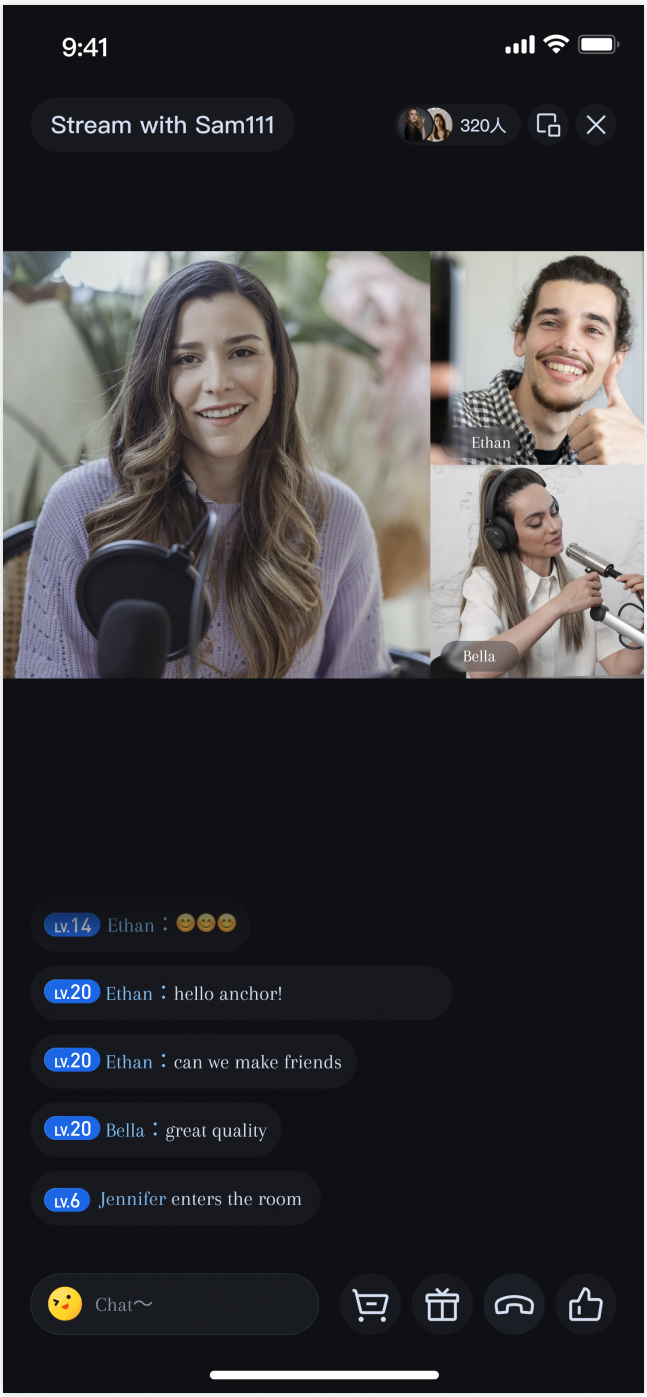
The `extInfo` of `TUIBarrage` is a Map, used for storing custom data.

Android

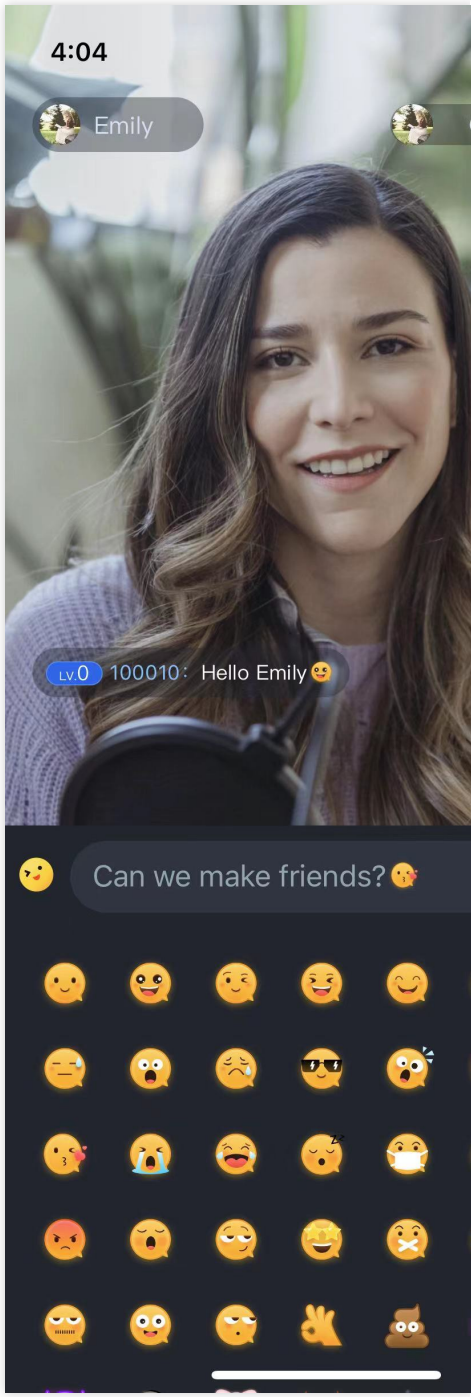
Last updated：2024-08-09 22:25:01

Overview

Live Chat feature supports the following functions: sending barrage messages, inserting custom messages, and custom message styles. Live Chat messages support emoji input, adding fun to the messages and making the interaction more enjoyable.



Display barrage



Send barrage

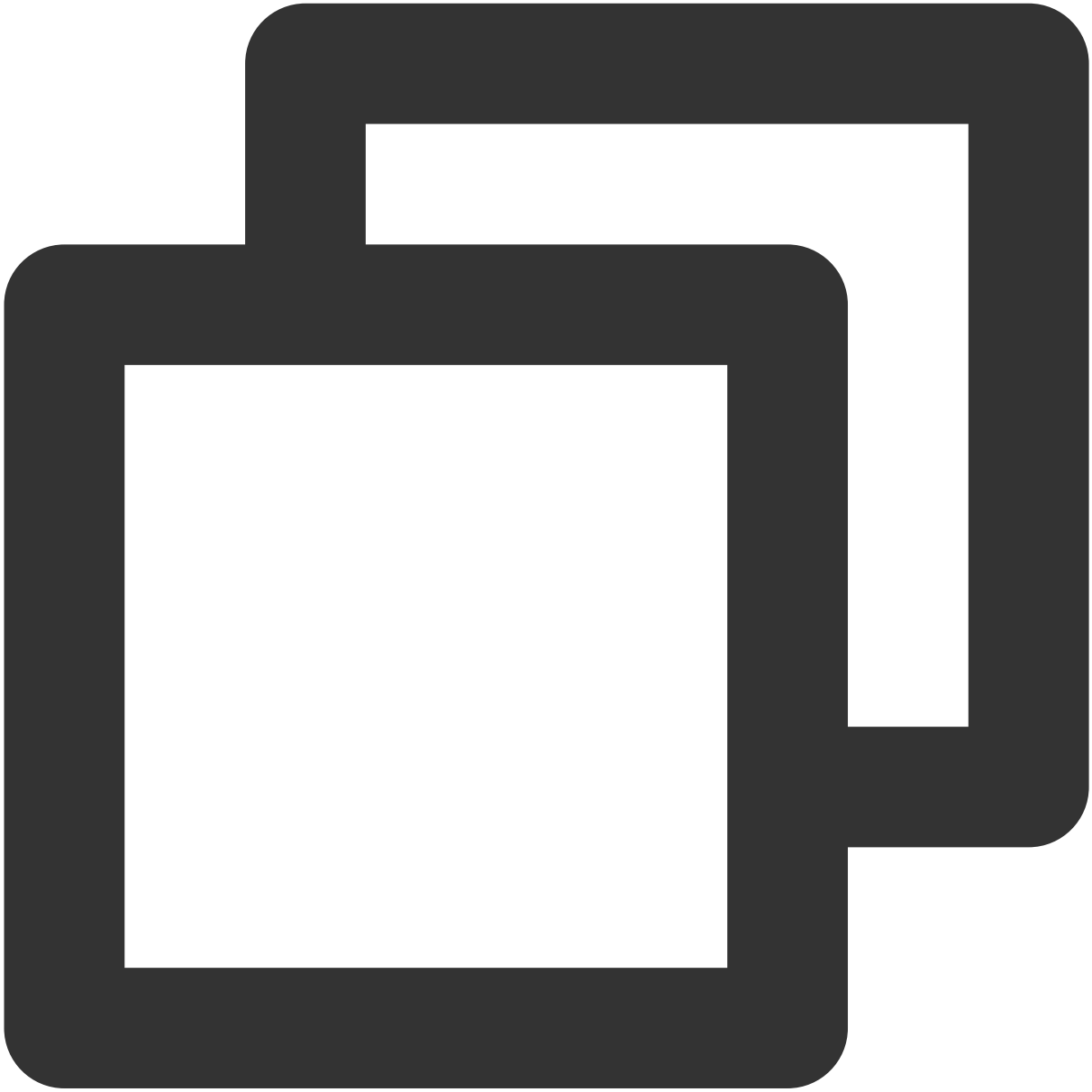
Note：

Support switching between **system keyboard** and **emoji keyboard**.

Integration

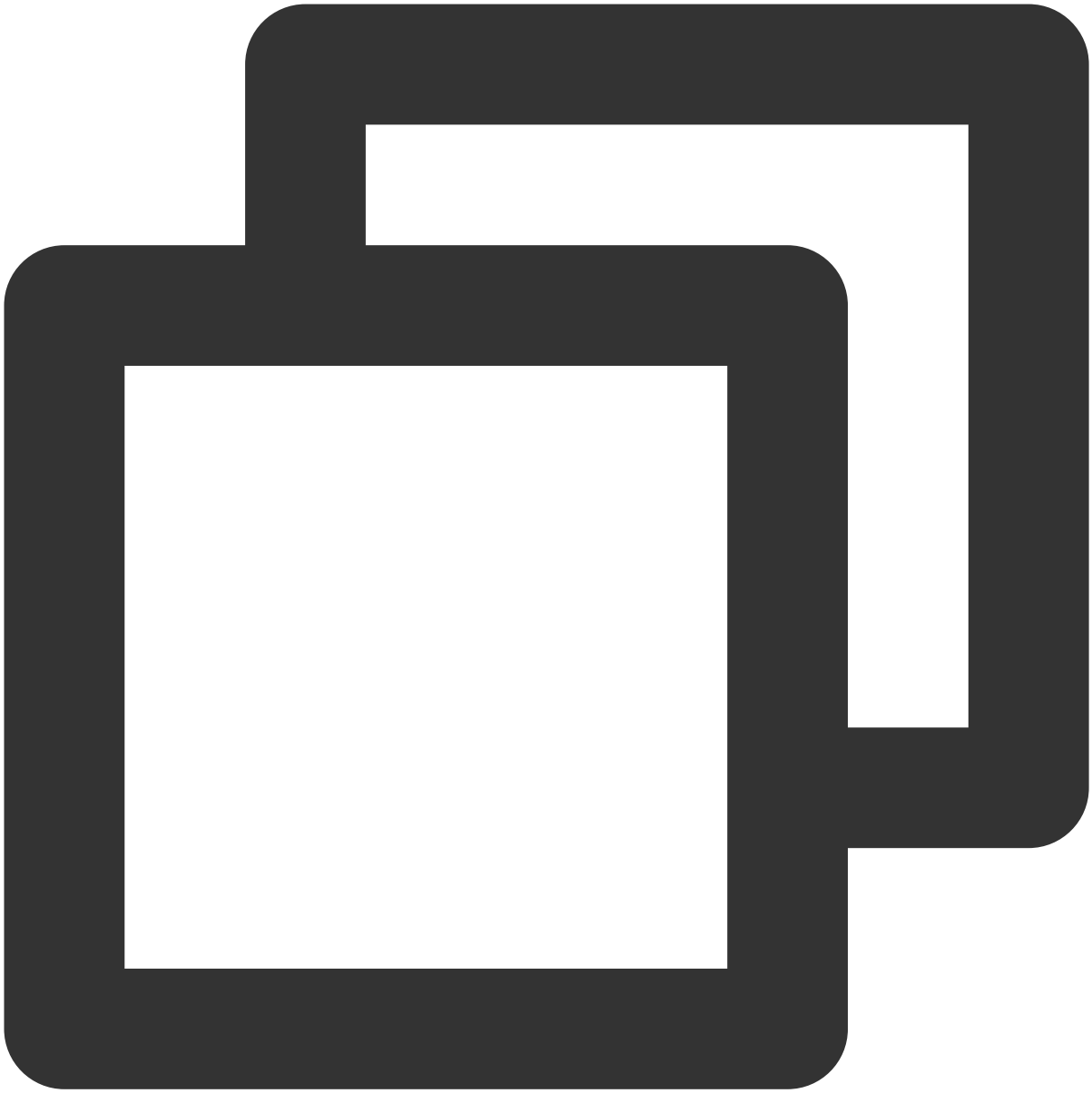
The barrage component mainly provides two Objects：

`TUIBarrageButton` : Clicking it can bring up the input interface.
`TUIBarrageDisplayView` : Used for displaying barrage messages.
In scenarios where barrage messages need to be sent, create a `TUIBarrageButton` , which can bring up the input interface when clicked:



```
TUIBarrageButton barrageButton = new TUIBarrageButton(mContext, roomId);
mBarrageButtonContainer.addView(barrageButton);
```

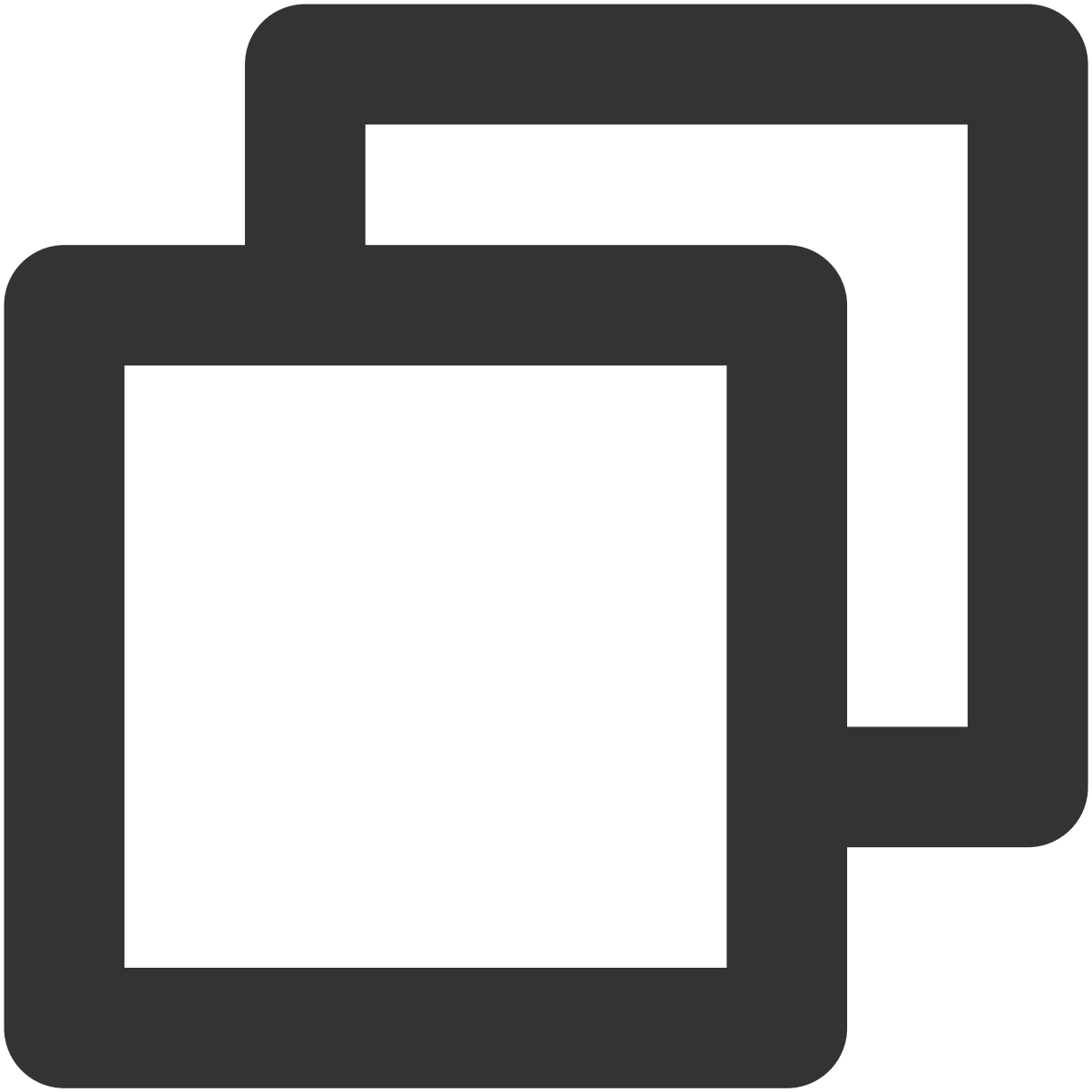
In scenarios where barrage messages need to be displayed, use `TUIBarrageDisplayView` to show the barrage messages:



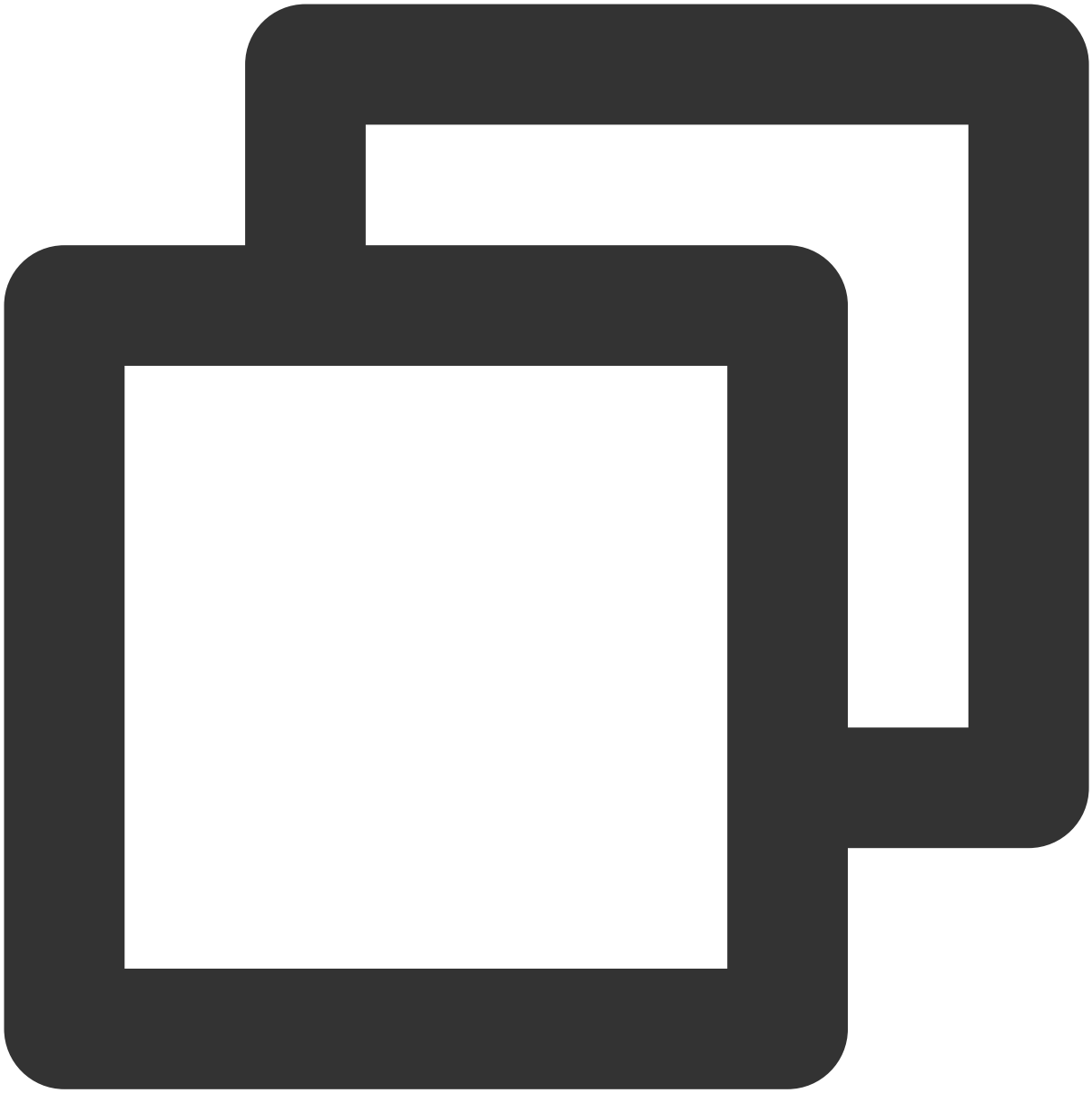
```
TUIBarrageDisplayView barrageDisplayView = new TUIBarrageDisplayView(mContext, room
mLayoutBarrageContainer.addView(barrageDisplayView);
```

Customize message style

The barrage display component `TUIBarrageDisplayView` provides `setAdapter` and `TUIBarrageDisplayAdapter` for customizing the pop-up message `Item` style:



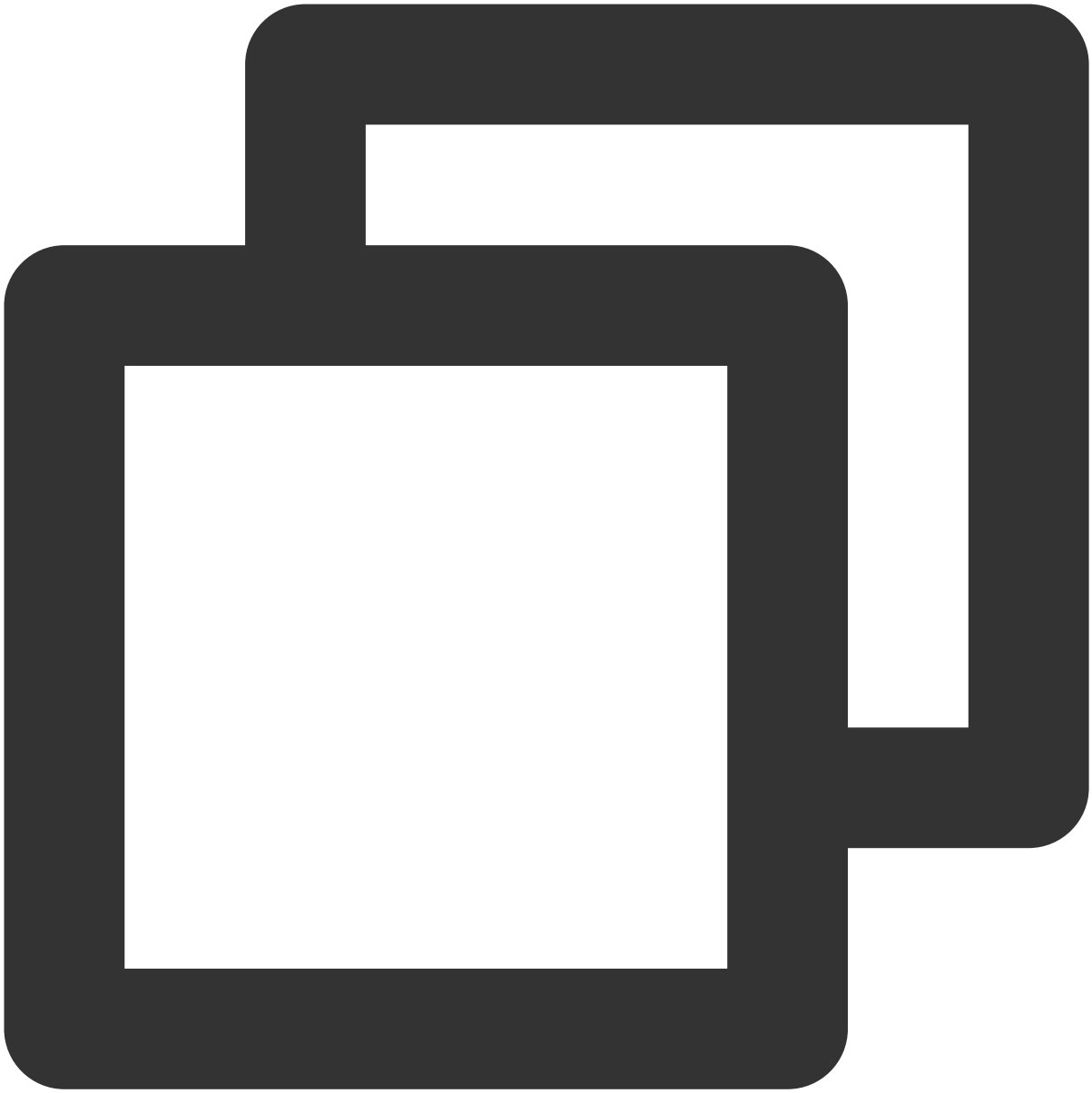
```
public interface TUIBarrageDisplayAdapter {
    RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType);

    void onBindViewHolder(RecyclerView.ViewHolder holder, TUIBarrage barrage);

    int getItemViewType(int position, TUIBarrage barrage);
}
```



```
public class GiftBarrageAdapter implements TUIBarrageDisplayAdapter {
    private final Context       mContext;

    public GiftBarrageAdapter(Context context) {
        mContext = context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewTyp
        if (viewType == GIFT_VIEW_TYPE_1) {
```

```
            // Handling of custom style 1
            LinearLayout ll = new LinearLayout(mContext);
            ll.addView(new TextView(mContext));
            return new GiftViewHolder(ll);
        }
        return null;
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder holder, TUIBarrage barrage
        if (holder instanceof GiftViewHolder) {
            GiftViewHolder viewHolder = (GiftViewHolder) holder;
            viewHolder.bind(barrage);
        }
    }

    @Override
    public int getItemViewType(int position, TUIBarrage barrage) {
        if (...) { // If the current barrage requires a custom Item style, return t
            return GIFT_VIEW_TYPE_1;
        }
        return 0; // 0 indicates that the default style is used
    }

    private static class GiftViewHolder extends RecyclerView.ViewHolder {

        public GiftViewHolder(View itemView) {
            super(itemView);
            // ...
        }

        public void bind(TUIBarrage barrage) {
            // ...
        }
    }
}

// set custom Adapter
barrageDisplayView.setAdapter(new GiftBarrageAdapter(mContext));
```

`TUIBarrage` is defined as follows:



```
public class TUIBarrage {
    public final TUIBarrageUser    user = new TUIBarrageUser(); //Sender
    public String                  content;                     //Sent content
    public HashMap<String, Object> extInfo = new HashMap<>();   //Expanded informat
}

public class TUIBarrageUser {
    public String userId;
    public String userName;
    public String avatarUrl;
```

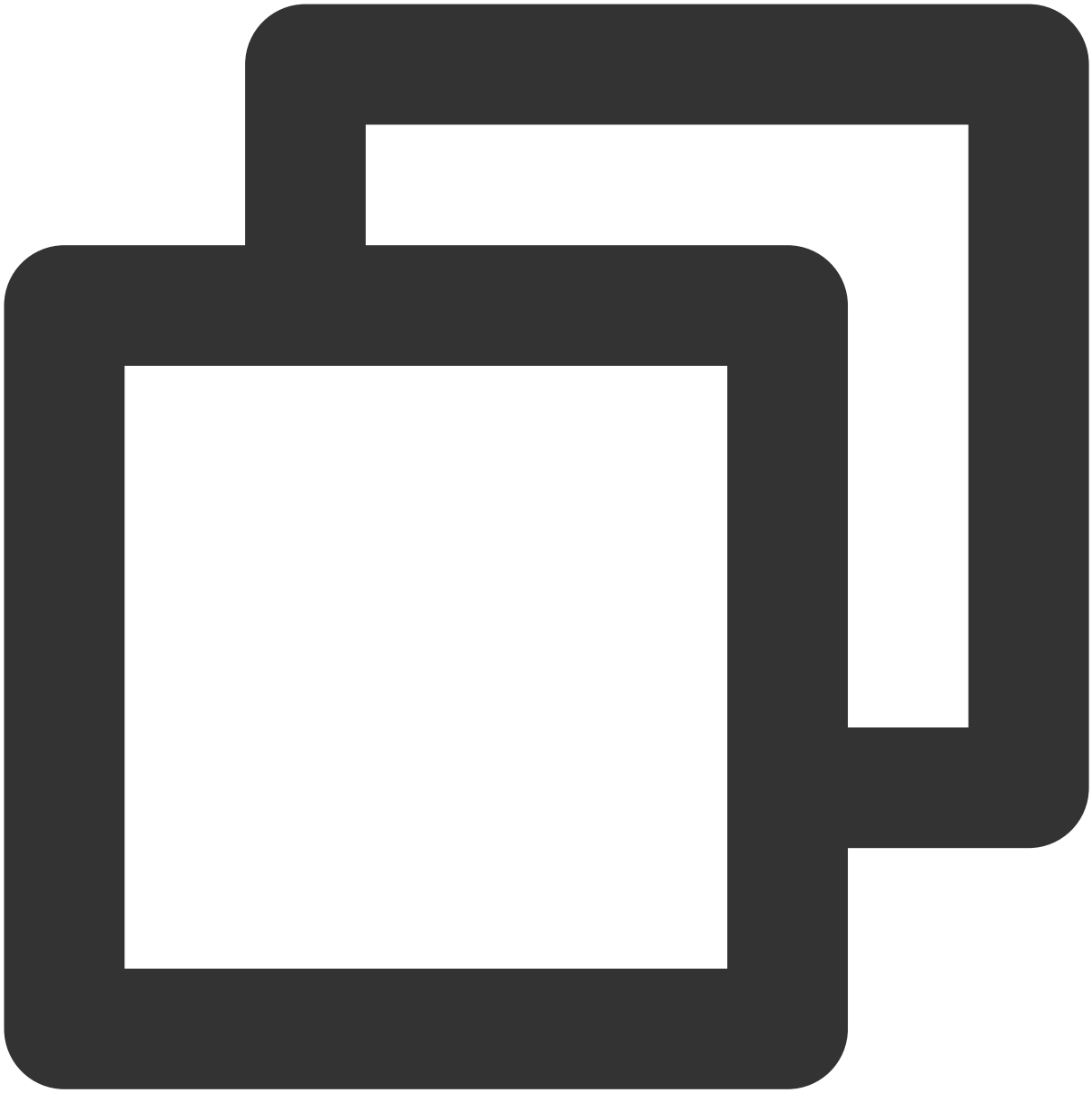
```
    public String level;
}
```

Note：

Supports multiple custom styles (specified by multiple return values through `getItemViewType`), 0 represents the default style.

InsertCustomMessage

The barrage display component `TUIBarrageDisplayView` provides the external interface method `insertBarrages` for inserting custom messages (in batches). Custom messages are usually used in combination with custom styles to achieve different display effects.



```
// Example: Insert a gift message in the barrage area.
TUIBarrage barrage = new TUIBarrage();
barrage.content = "gift";
barrage.user.userId = sender.userId;
barrage.user.userName = sender.userName;
barrage.user.avatarUrl = sender.avatarUrl;
barrage.user.level = sender.level;
barrage.extInfo.put(Constants.GIFT_VIEW_TYPE, GIFT_VIEW_TYPE_1);
barrage.extInfo.put(GIFT_NAME, barrage.giftName);
barrage.extInfo.put(GIFT_COUNT, giftCount);
barrage.extInfo.put(GIFT_ICON_URL, barrage.imageUrl);
```

```
barrage.extInfo.put(GIFT_RECEIVER_USERNAME, receiver.userName);
barrageDisplayView.insertBarrages(barrage);
```

Note：

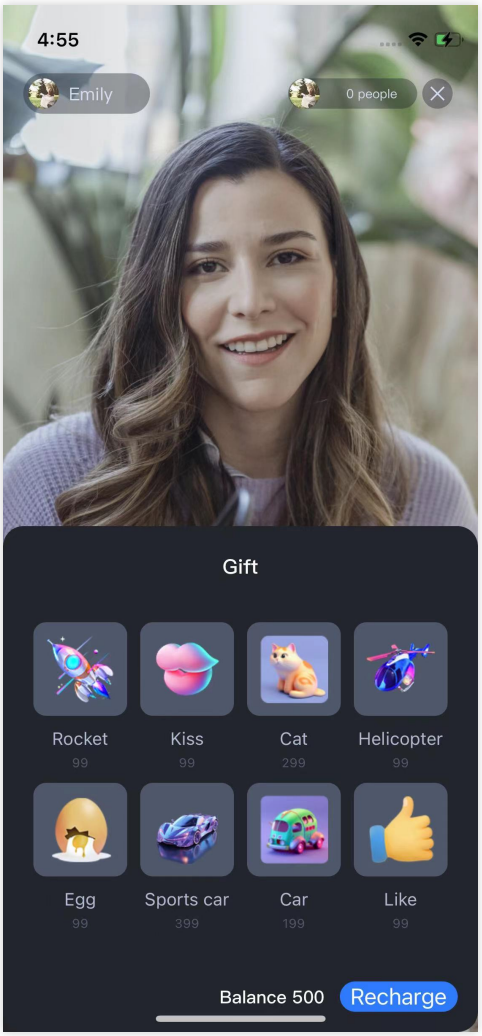
The `extInfo` of `TUIBarrage` is a Map, used for storing custom data.

Interactive Gifts (TUILiveKit)
iOS

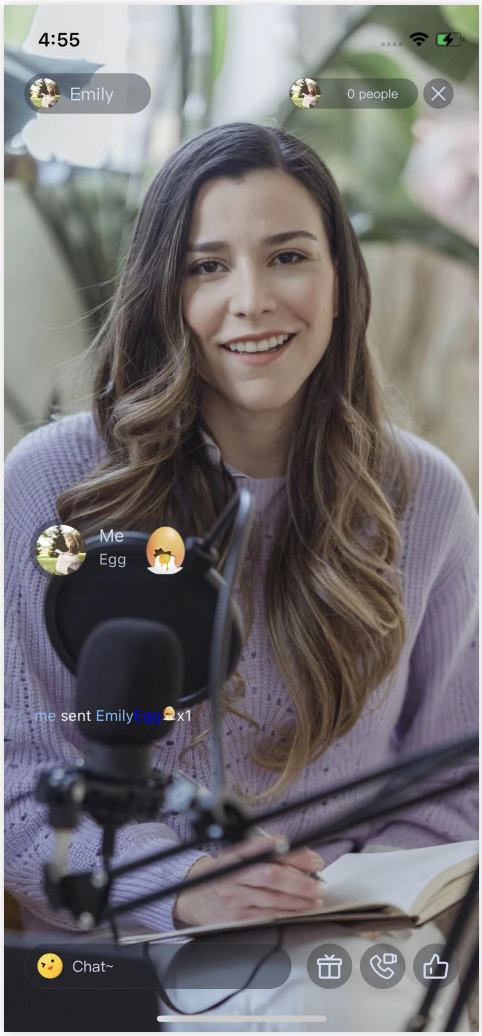
Last updated：2024-07-10 16:31:15

The interactive gift component is a virtual gift interaction platform designed to add more fun to users' social experiences. With this component, users can send virtual gifts to their favorite live streamers to show their appreciation, love, and support.
The interactive gift component supports setting **gift materials**, **displaying balance**, **playing ordinary gifts** and **full-screen gifts**, and **adding a recharge button**, etc.

Overview



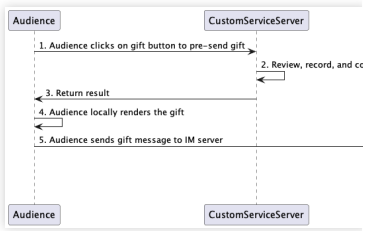
Display Gifts

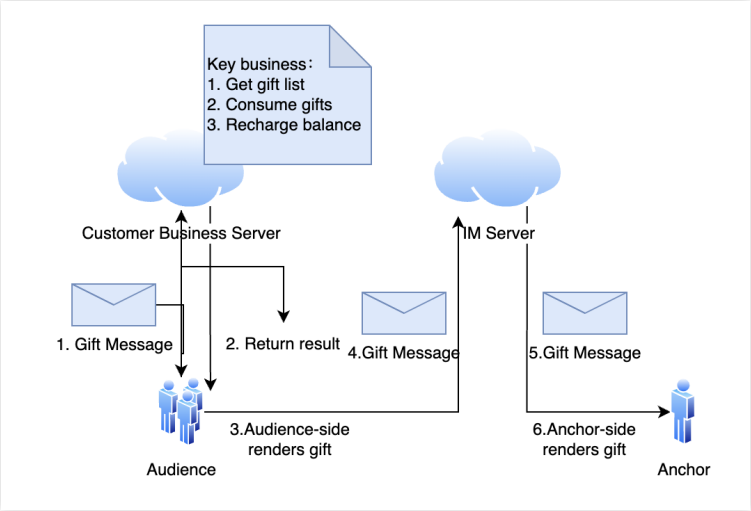


Play normal gift

Pl

Gift System



	
Gift system`s Structure diagram	Gift system`s Sequence diagram

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

1. After billing, the sender directly sees that XXX sent XXX gifts (to ensure that the sender sees the gifts he sent, and the abandonment policy may be triggered when the message volume is large).

2. After the billing is settled, call the GiftListView.sendGift to send a message to cancel the gift.

Integration

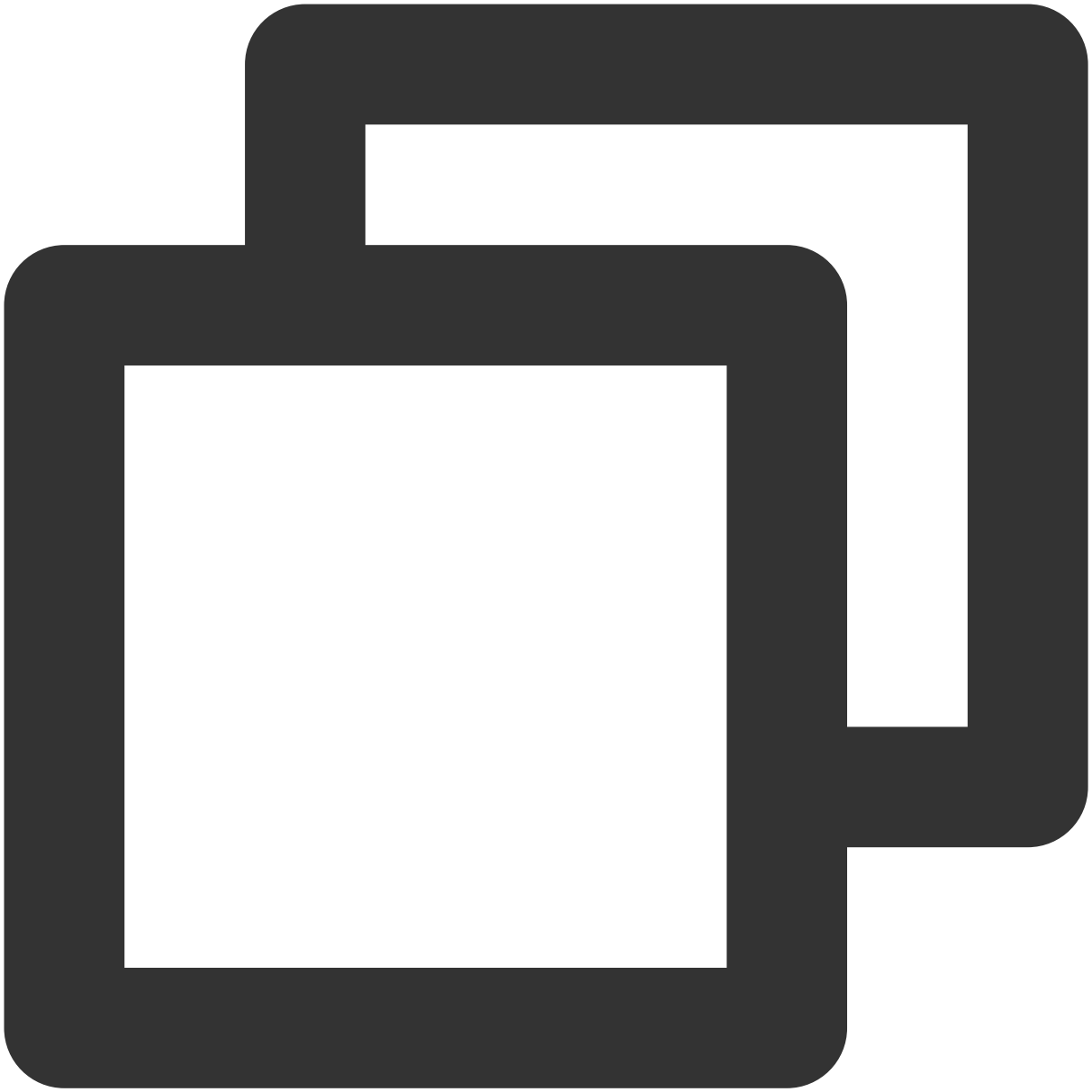
The gift component mainly provides 2 objects:

`TUIGiftListView` : A gift panel that presents the gift list, sends gifts, and recharges.

`TUIGiftPlayView` : A panel that plays gifts and automatically listens to gift messages.

Set gift materials

The gift panel component `TUIGiftListView` provides the `setGiftList` interface, which can be used to set gift materials.



```
let giftListView: TUIGiftListView = TUIGiftListView(groupId: xxx) //generator giftL
let giftList: [TUIGift] = ... //you can change gift materials here
giftListView.setGiftList(giftList) //set gift materials of giftListPanleView
```

Note：

The parameters and descriptions of `TUIGift` are as follows:

`giftId: String`： Gift ID

`giftName: String`： Gift Name

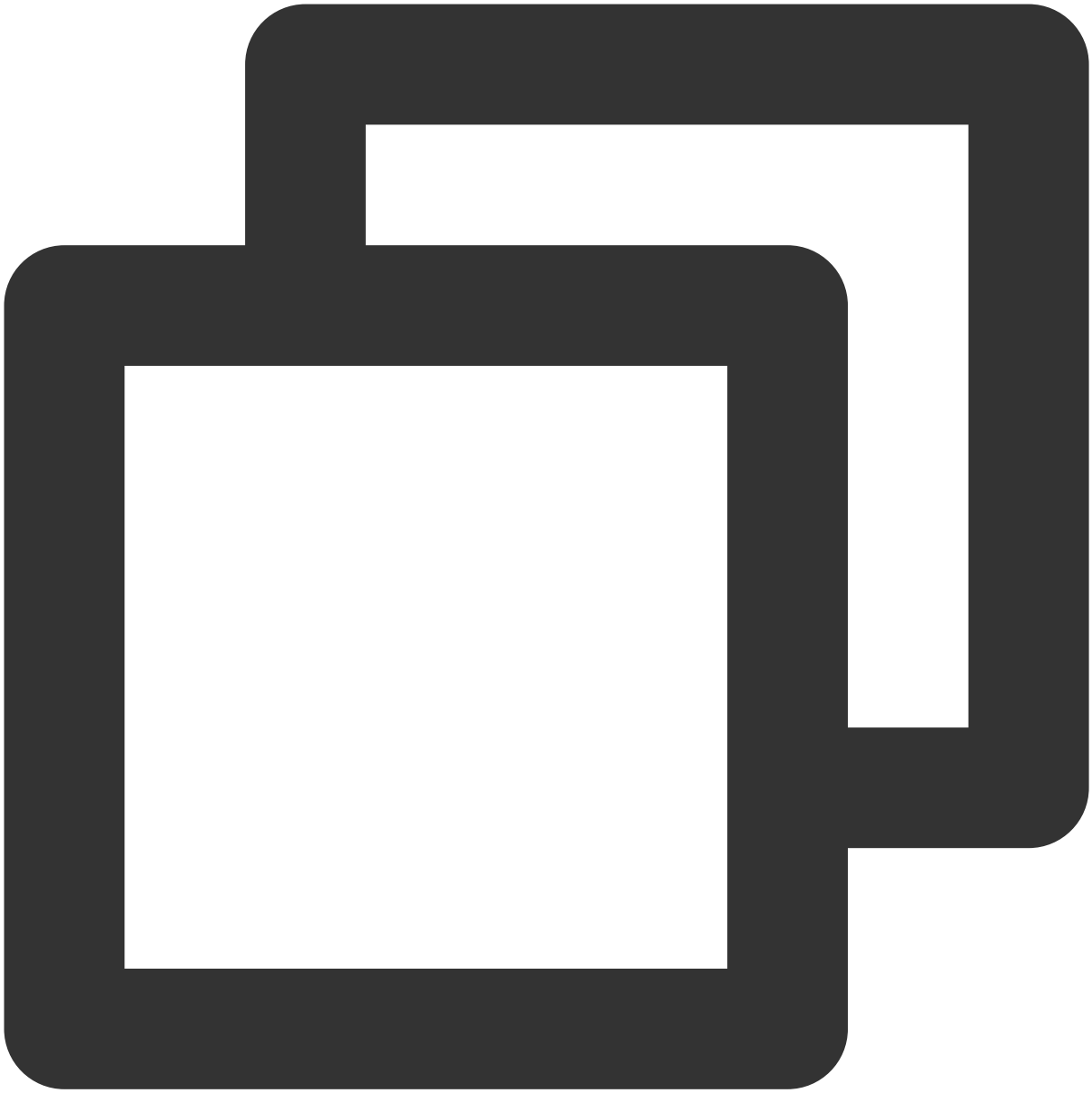
`imageUrl: String`： Image displayed on the gift panel

`animationUrl: String`: SVGA animation URL
`price: Int`: Gift Price
`extInfo: [String: AnyCodable]`: Custom extension information

The interactive gift component supports setting your own gift materials. If the `animationUrl` is empty, the gift playing effect will be an ordinary play, and the content played will be the image linked by the imageUrl. If the `animationUrl` is not empty, the playing effect will be a full-screen play, and the content played will be the corresponding svga animation.

Send gift

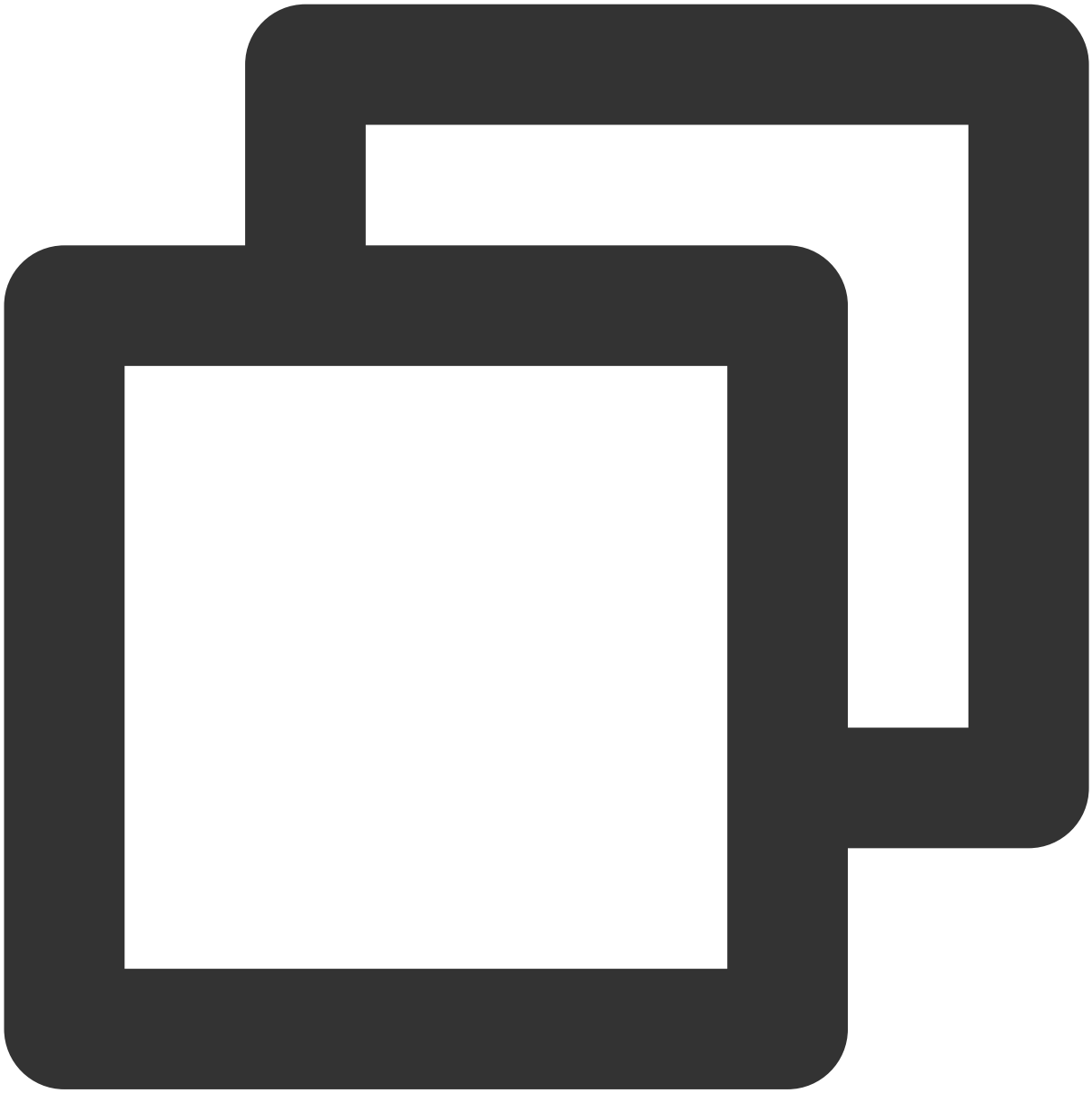
Implement the `onSendGift` delegate function in the `TUIGiftListViewDelegate` of `TUIGiftListView` to get the gift count and gift information. After preprocessing, you can call the `sendGift` function of `TUIGiftListView` for the actual sending of gifts.



```
giftListView.delegate = self

extension ViewController: TUIGiftListViewDelegate {
    func onSendGift(giftListView view: TUIGiftListView, giftModel: TUIGift, giftCou
        //...Here is the preprocessing, such as verifying the current user's balanc
        let receiver = TUIGiftUser()
        //...Set the gift recipient information here.
        view.sendGift(giftModel: giftModel, giftCount: giftCount, receiver: receive
    }
}
```

ReceiveGift

The gift display component `TUIGiftPlayView` will receive and play gift messages by itself.

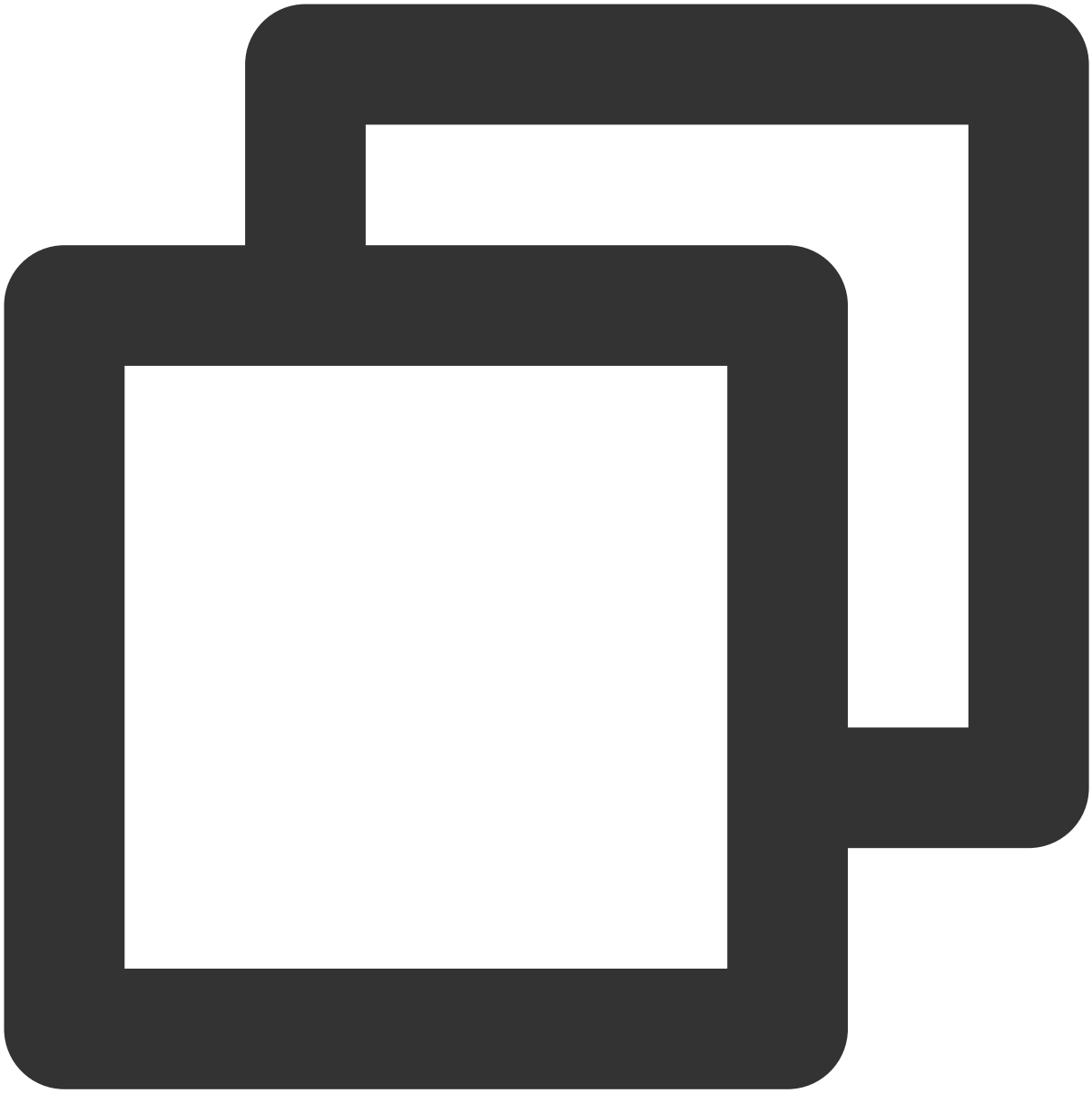


```
let giftDisplayView: TUIGiftPlayView = TUIGiftPlayView(groupId:xxx)
```

Note：

`TUIGiftPlayView` requires full-screen integration.

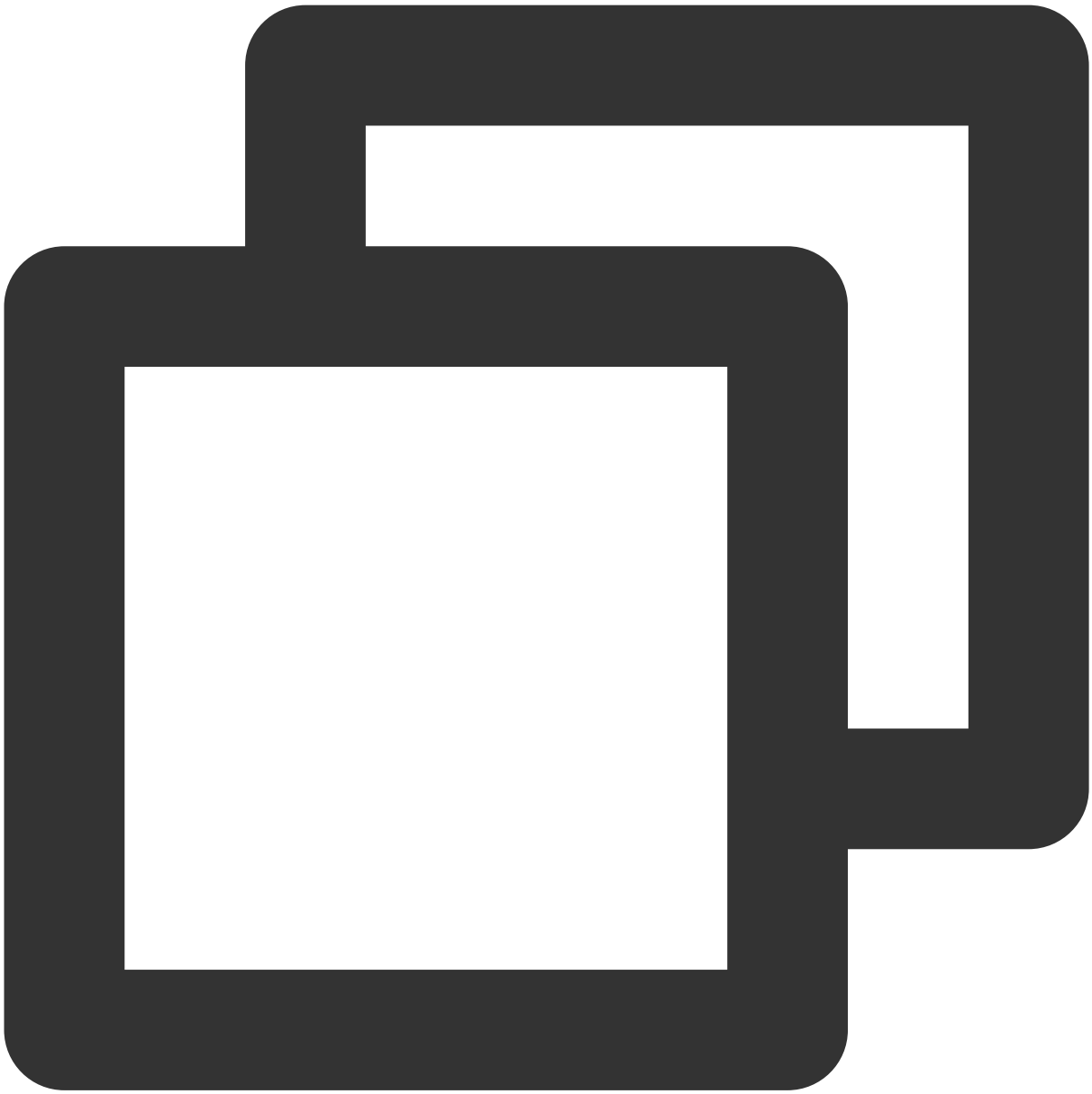
If you need to get the callback information of receiving gifts, you can implement the `giftPlayView:onPlayGift` delegate function in the `TUIGiftPlayViewDelegate` of `TUIGiftPlayView` .



```
extension ViewController: TUIGiftPlayViewDelegate {
    func giftPlayView(_ giftPlayView: TUIGiftPlayView, onPlayGift gift: TUIGift, gi
     //...You can handle this on your own here.
    }
}
```

Set balance

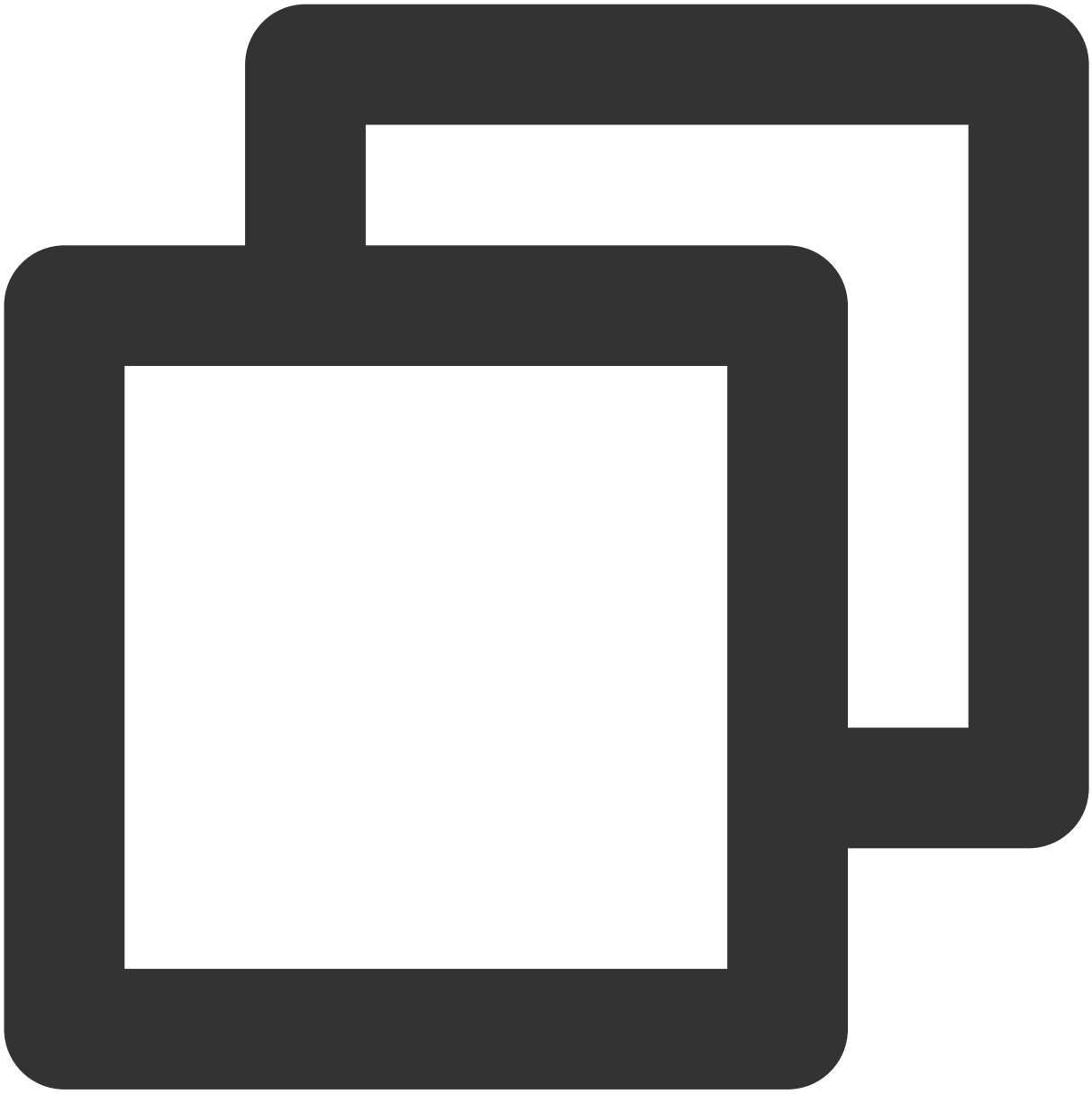
The gift panel component `TUIGiftListView` provides the `setBalance` interface, which can be used to set the balance value displayed on the gift panel.



```
giftListView.setBalance(xxx)
```

Recharge

Implement the `onRecharge` delegate function in the `TUIGiftListViewDelegate` of `TUIGiftListView` , which can be used to receive the click event of the recharge button thrown by the gift display panel, and connect to your own recharge system here.



```
extension ViewController: TUIGiftListViewDelegate {
    func onRecharge(giftListView view: TUIGiftListView) {
        //...This can be used to connect to your own recharge system. After the rec
        //you can call view.setBalance(xxx) to set the balance display of the gift
    }
}
```

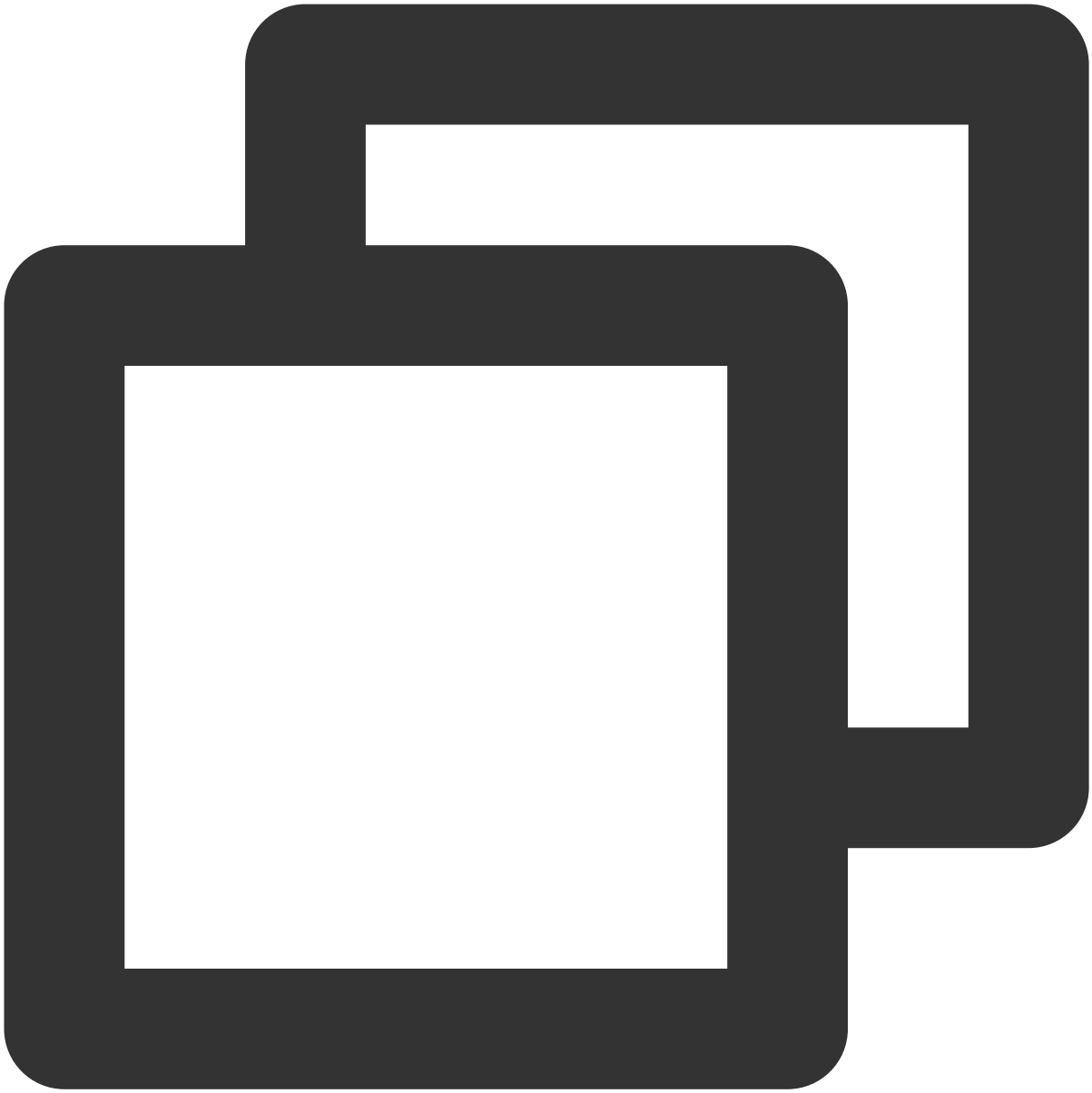
Note：

The gift balance is a concept of virtual currency, not real money.

The gift recharge logic is implemented externally, and customers can connect to their own recharge system. After the recharge is completed, the gift balance is updated.

Billing statistics

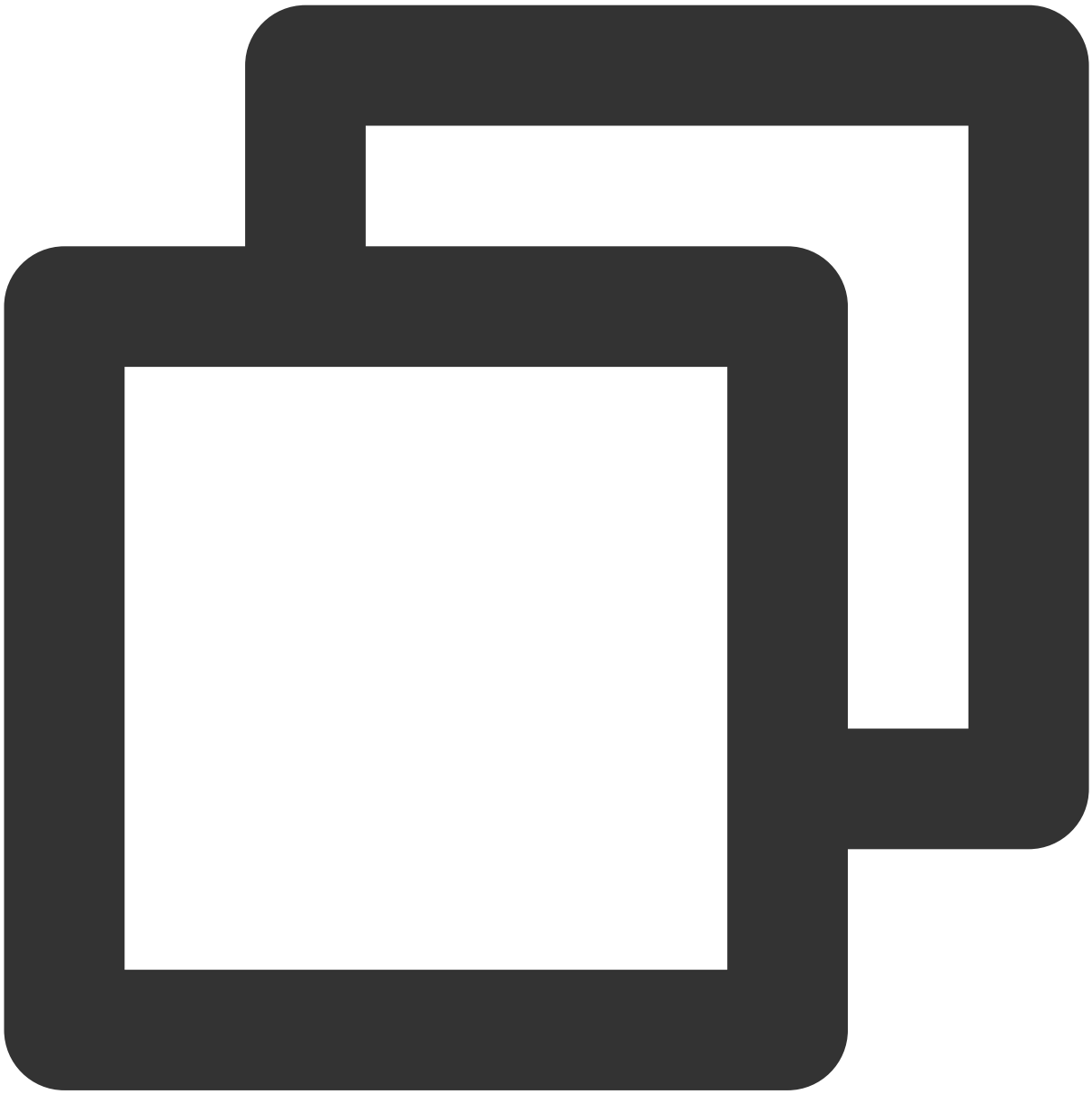
Implement the `onSendGift` delegate function in the `TUIGiftListViewDelegate` of `TUIGiftListView` , connect to the customer's own business server, complete the balance verification, gift billing, and consumption statistics, and then call the `sendGift` of `TUIGiftListView` to send the gift message.



```
giftListView.delegate = self

extension ViewController: TUIGiftListViewDelegate {
    func onSendGift(giftListView view: TUIGiftListView, giftModel: TUIGift, giftCou
        //...Connect to the customer's own business server here to complete balance
        let receiver = TUIGiftUser()
        //...Set the gift recipient information here.
        view.sendGift(giftModel: giftModel, giftCount: giftCount, receiver: receive
    }
}
```

Customize giftList

Modify the gift list on the audience's gift panel:



```
// Source code path：TUILiveKit/Source/LiveRoom/View/Audience/Component/AudienceLiv

private lazy var giftPanelView: TUIGiftListView = {
    let view = TUIGiftListView(groupId: liveRoomInfo.roomId.value)
    giftCloudServer.queryGiftInfoList { [weak self] error, giftList in
        guard let self = self else { return }
        DispatchQueue.main.async {
```

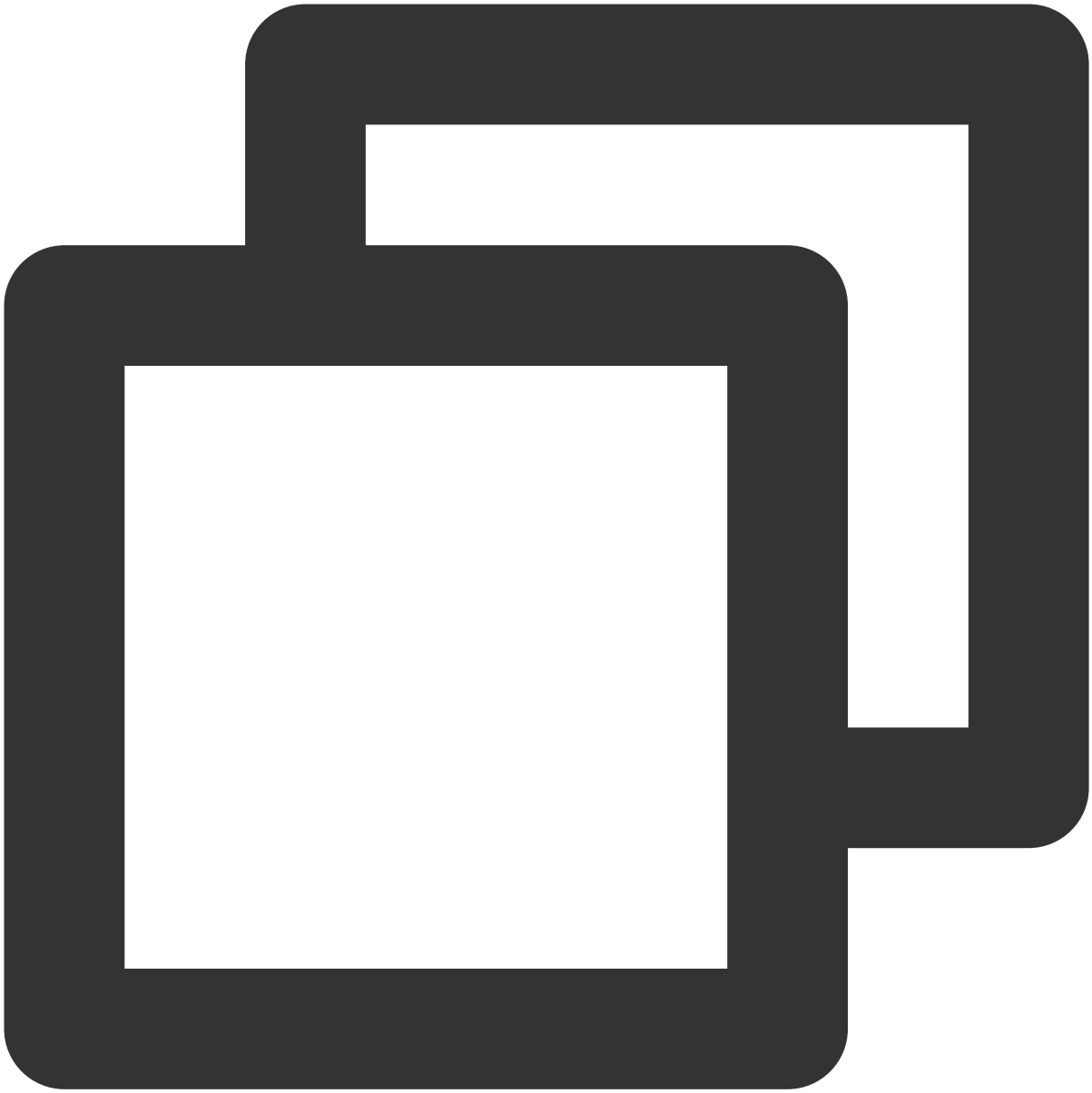
```
            if error == .noError {
                view.setGiftList(giftList)
            } else {
                self.makeToast("query gift list error, code = \\(error)")
            }
        }
    }
    return view
}()
```

Note：

Customers implement the logic of `giftCloudServer.queryGiftInfoList` on their own, get a custom gift list `[TUIGift]` , and set the gift list through `view.setGiftList` .

The animationUrl of the gift is required to be a SVGA animation.

Customize giftPanel`s balance



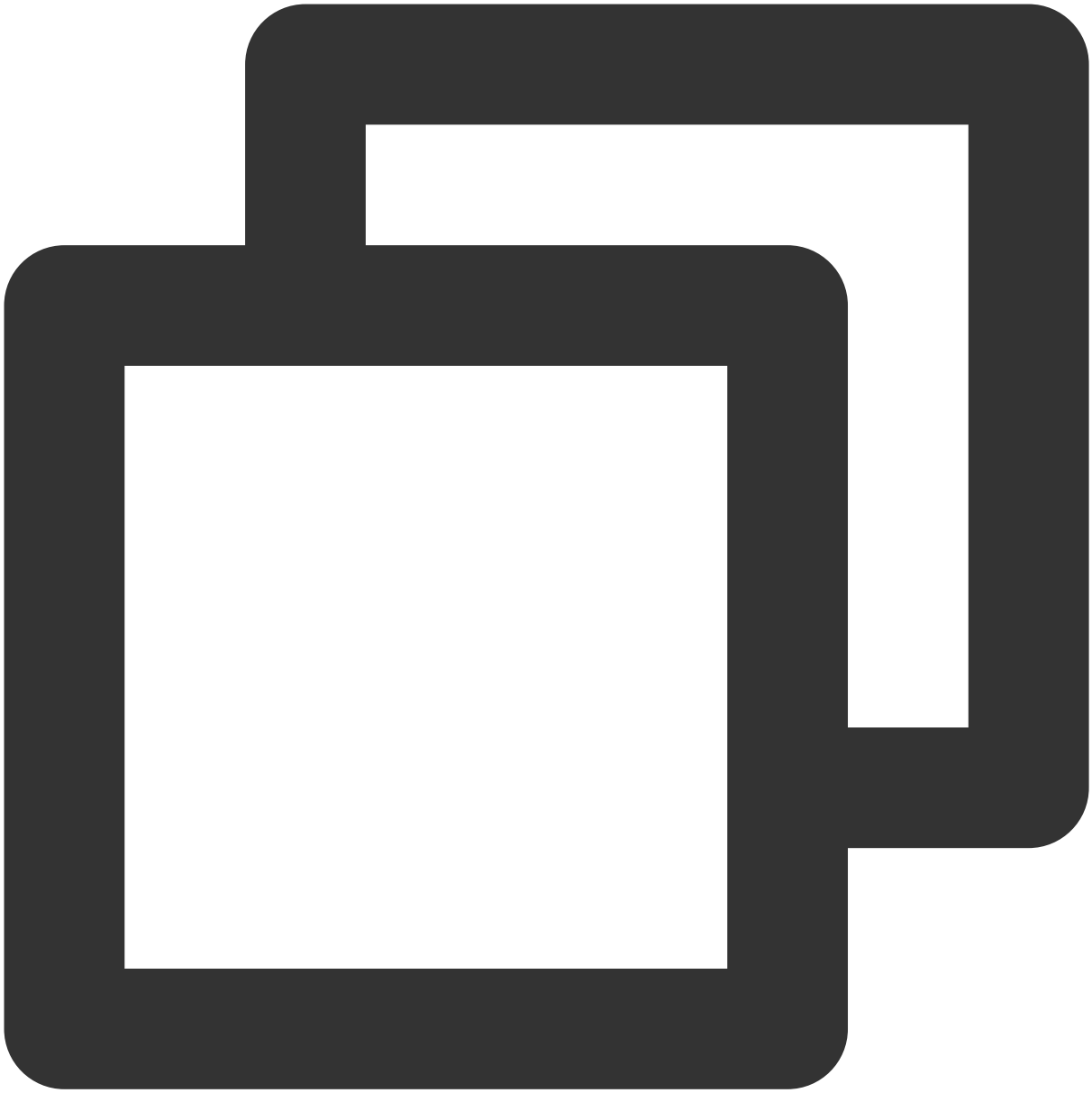
```
// Source code path：TUILiveKit/Source/LiveRoom/View/Audience/Component/AudienceLiv

private lazy var giftPanelView: TUIGiftListView = {
    let view = TUIGiftListView(groupId: liveRoomInfo.roomId.value)
    giftCloudServer.queryBalance { [weak self] error, balance in
        guard let self = self else { return }
        DispatchQueue.main.async {
            if error == .noError {
                view.setBalance(balance)
            } else {
                self.makeToast("query balance error, code = \\(error)")
```

```
                }
            }
        }
        return view
    }()
```

Note：

Customers implement the logic of `giftCloudServer.queryBalance` on their own, obtain the gift balance, and update the gift balance through `view.setBalance` .

Customize gift consumption logic



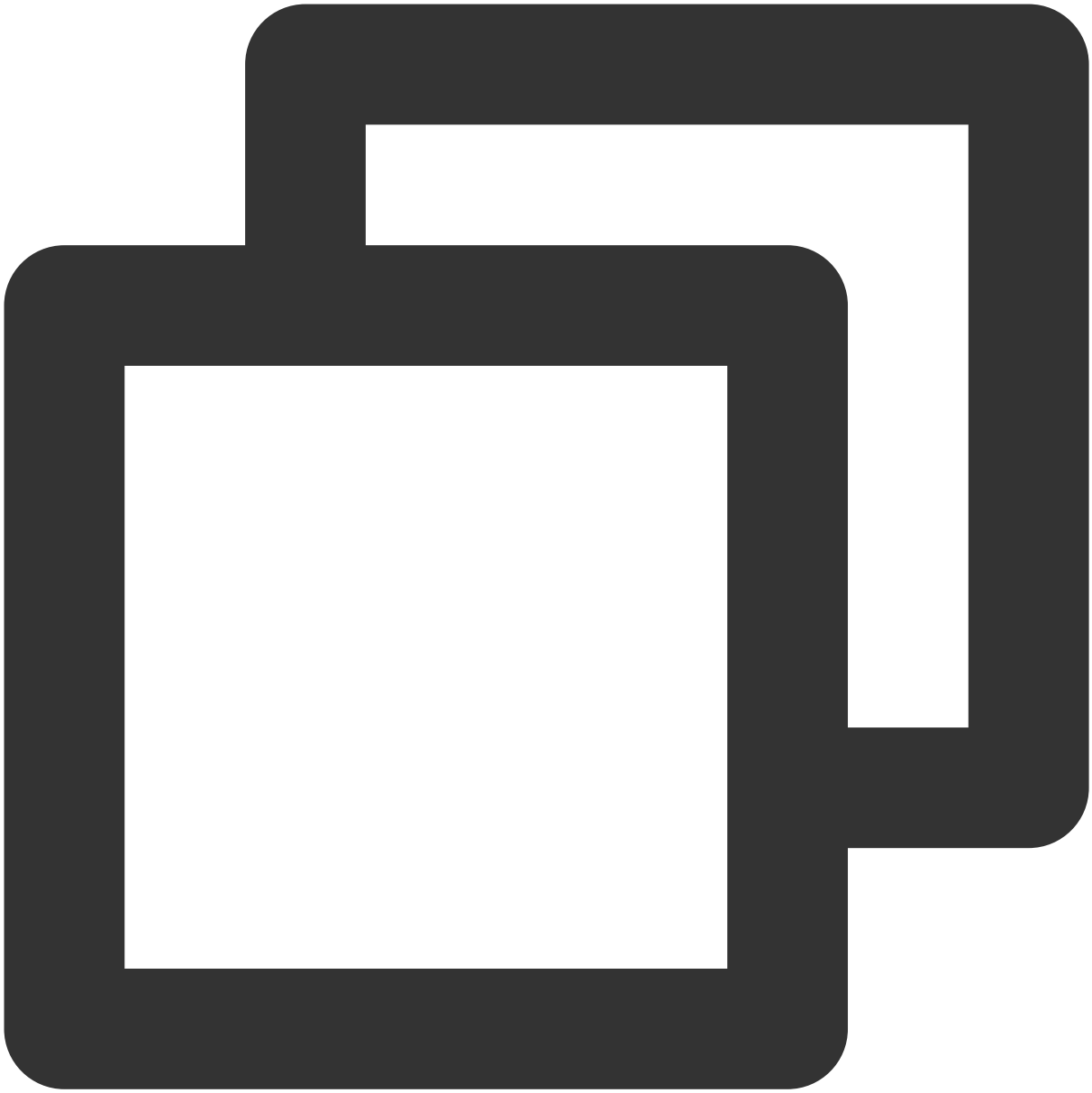
```
// Source code path：TUILiveKit/Source/LiveRoom/View/Audience/Component/AudienceLiv

func onSendGift(giftListView view: TUIGiftListView, giftModel: TUIGift, giftCount:
    let receiver = TUIGiftUser()
    receiver.userId = liveRoomInfo.anchorInfo.value.userId
    receiver.userName = liveRoomInfo.anchorInfo.value.name.value
    receiver.avatarUrl = liveRoomInfo.anchorInfo.value.avatarUrl.value
    receiver.level = "0"
    giftCloudServer.sendGift(sender: TUILogin.getUserID() ?? "",

                             receiver: receiver.userId,
```

```
                                giftModel: giftModel,

                                giftCount: giftCount) { [weak self] error, balance in
            guard let self = self else { return }
            if error == .noError {
                view.sendGift(giftModel: giftModel, giftCount: giftCount, receiver: rec
                view.setBalance(balance)
            } else {
                self.makeToast(.balanceInsufficientText)
            }
        }
    }
```

Note：

Customers implement the logic of `giftCloudServer.sendGift` on their own. The main logic is to first connect to the customer's own business server to verify the balance, and after the verification is passed, the server will charge and count the consumption records, and finally call back the result to the client. After receiving the successful callback, the client sends the gift message through the `sendGift` of the `GiftListView`, and then updates the gift balance through `setBalance`.

Customize the loading gift animation and play



```
// Source code path:
// TUILiveKit/Source/LiveRoom/View/Audience/Component/AudienceLivingView.swift
// TUILiveKit/Source/LiveRoom/View/Anchor/Living/AudienceLivingView.swift

func giftPlayView(_ giftPlayView: TUIGiftPlayView, onPlayGiftAnimation gift: TUIGif
    giftCacheService.request(urlString: gift.animationUrl) { error, data in
        guard let data = data else { return }
        if error == 0 {
            DispatchQueue.main.async {
                giftPlayView.playGiftAnimation(animationData: data)
```

```
                }
            }
        }
    }
```

Note:

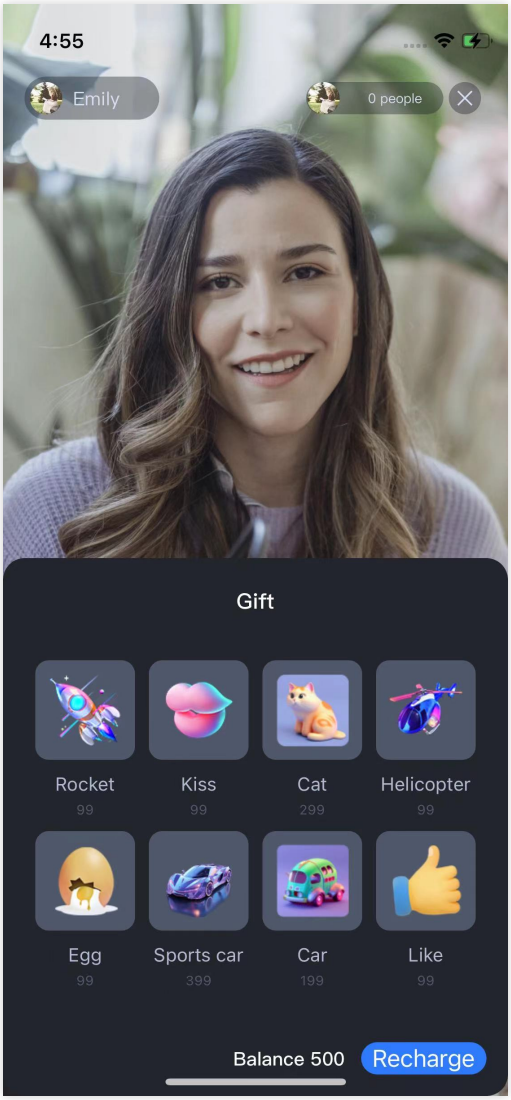
Customers implement the logic of `giftCacheService.request` on their own. Lexter, loads the animation to get the data (`Data` type), and then plays the gift animation through `playGiftAnimation` of `TUIGiftPlayView` .

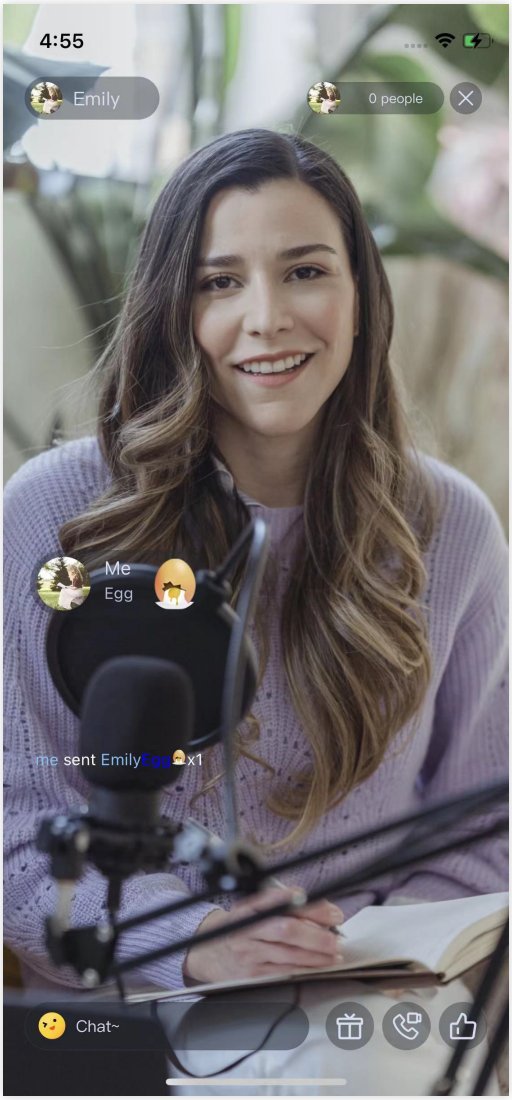
Android

Last updated：2024-07-10 16:31:15

The interactive gift component is a virtual gift interaction platform designed to add more fun to users' social experiences. With this component, users can send virtual gifts to their favorite live streamers to show their appreciation, love, and support.
The interactive gift component supports setting **gift materials**, **displaying balance**, **playing ordinary gifts** and **full-screen gifts**, and **adding a recharge button**, etc.

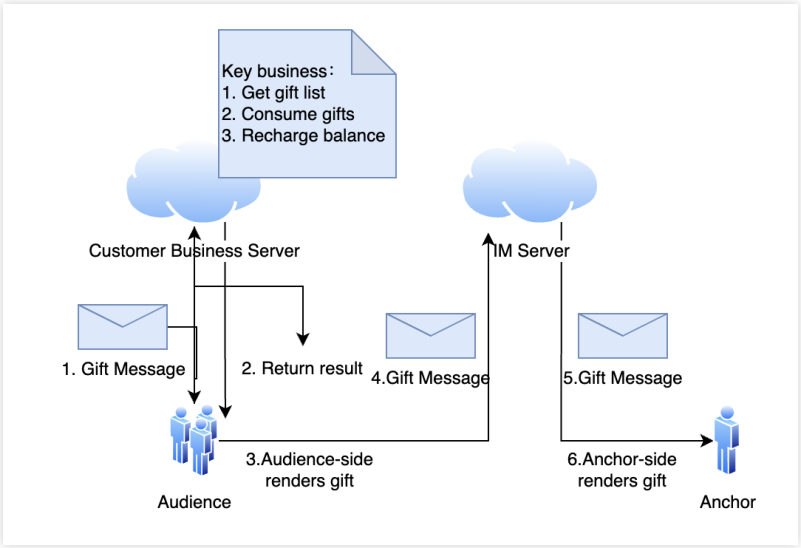
Overview

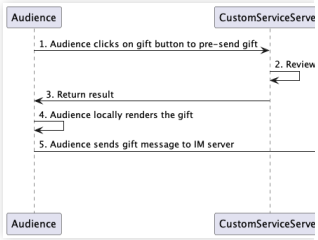




Display Gifts	Play normal gift

Gift System





Gift system`s Structure diagram	Gift system`s Sequence diagram

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

1. After billing, the sender directly sees that XXX sent XXX gifts (to ensure that the sender sees the gifts he sent, and the abandonment policy may be triggered when the message volume is large).

2. After the billing is settled, call the GiftListView.sendGift to send a message to cancel the gift.

Integration

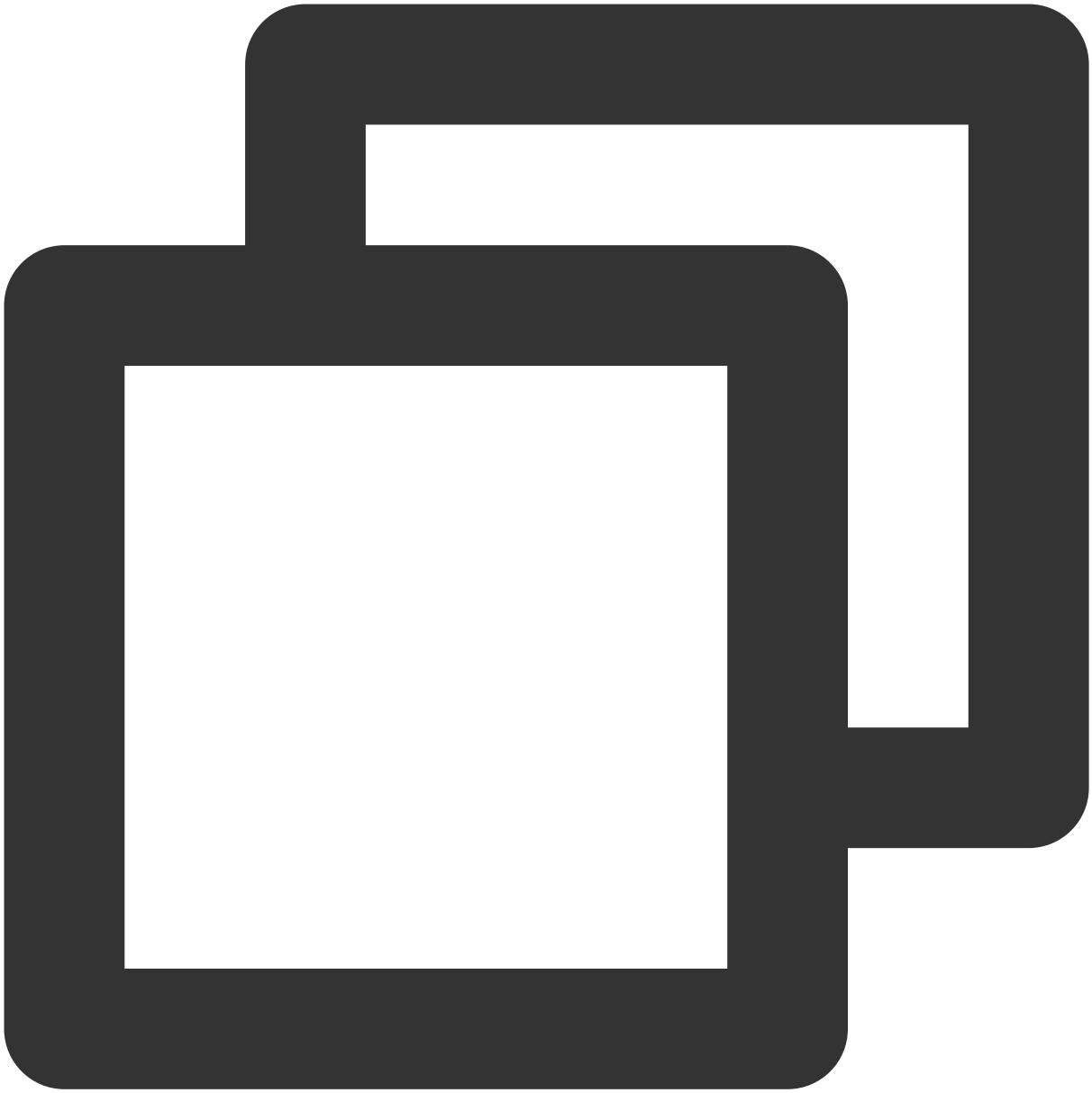
The gift component mainly provides 2 objects:

`TUIGiftListView` : A gift panel that presents the gift list, sends gifts, and recharges.

`TUIGiftPlayView` : A panel that plays gifts and automatically listens to gift messages.

Set gift materials

The gift panel component `TUIGiftListView` provides the `setGiftList` interface, which can be used to set gift materials.



```
TUIGiftListView giftListView = new TUIGiftListView(mContext, roomId); //generator g
List<TUIGift> giftList = new ArrayList<>() //you can change gift materials here
giftListView.setGiftList(giftList) //set gift materials of giftListPanleView
```

Note：

The parameters and descriptions of `TUIGift` are as follows:

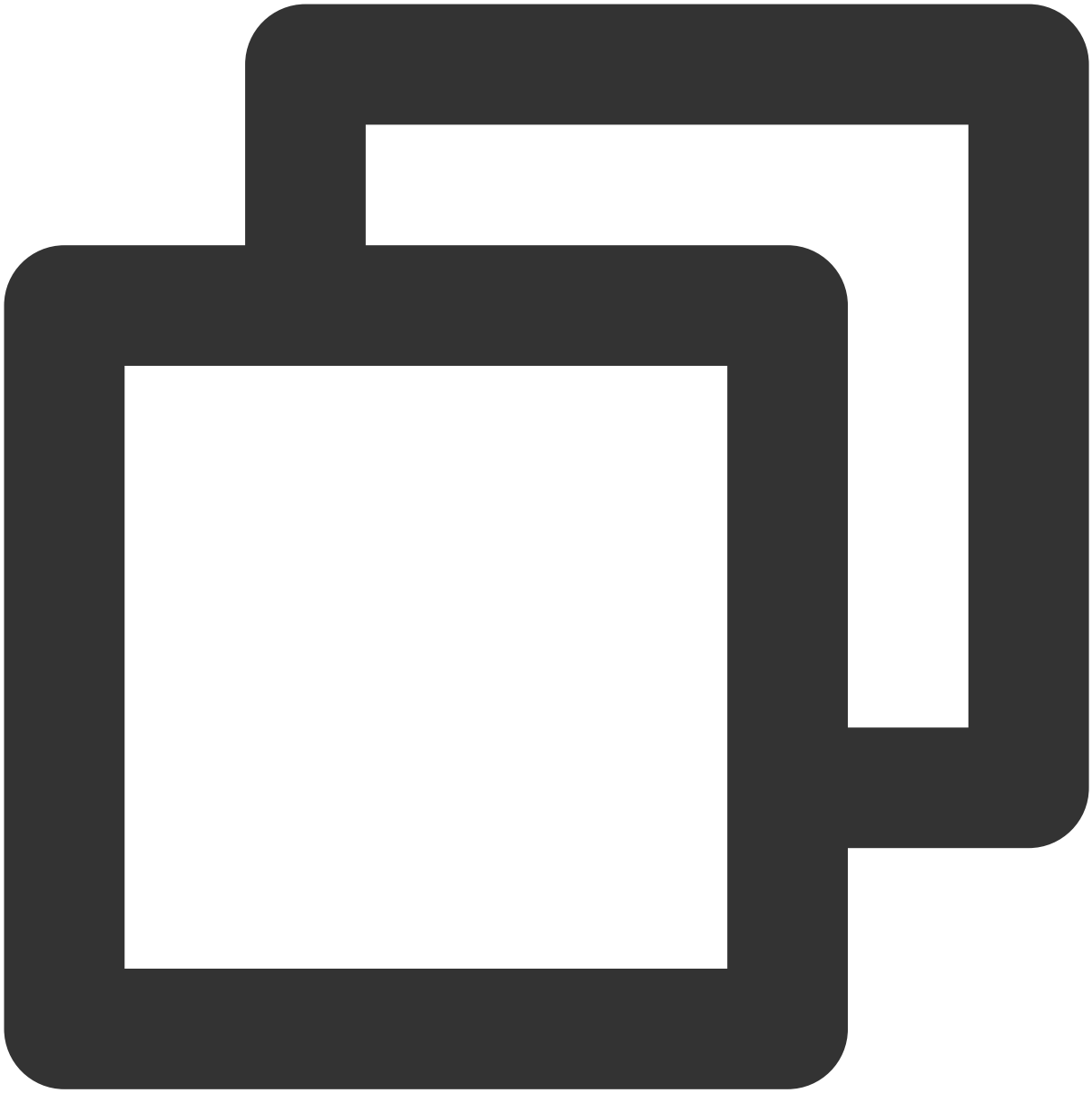
`giftId: String`	Gift ID
`giftName: String`	Gift Name
`imageUrl: String`	Image displayed on the gift panel

`animationUrl: String`	SVGA animation URL
`price: Int`	Gift Price
`extInfo: <String, Object>`	Custom extension information

The interactive gift component supports setting your own gift materials. If the `animationUrl` is empty, the gift playing effect will be an ordinary play, and the content played will be the image linked by the imageUrl. If the `animationUrl` is not empty, the playing effect will be a full-screen play, and the content played will be the corresponding svga animation.

Send gift

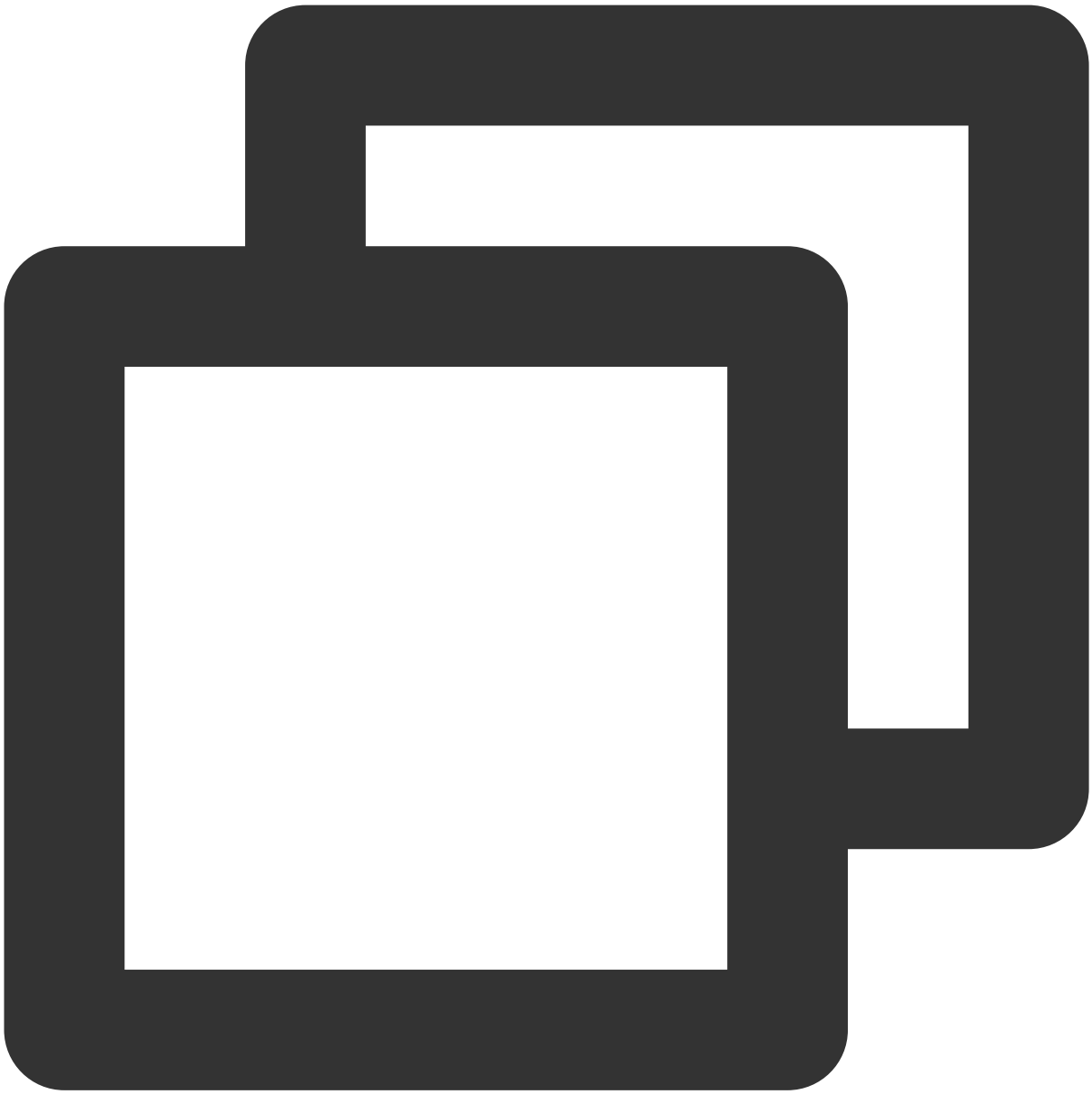
Implement the `onSendGift` callback in the `OnGiftListener` of `TUIGiftListView`, get the number of gifts and gift information, after preprocessing, you can call the `sendGift` function of `TUIGiftListView` for the actual sending of gifts.



```
public void onSendGift(TUIGiftListView giftListView, TUIGift gift, int giftCount) {
    //...This operation is preprocessing, such as verifying the balance of the curre
    TUIGiftUser receiver = new TUIGiftUser();
    //...Set the gift receiver information here
    giftListView.sendGift(gift, giftCount, receiver);
}
```

Receive Gift

The gift display component `TUIGiftPlayView` will receive and play gift messages by itself.

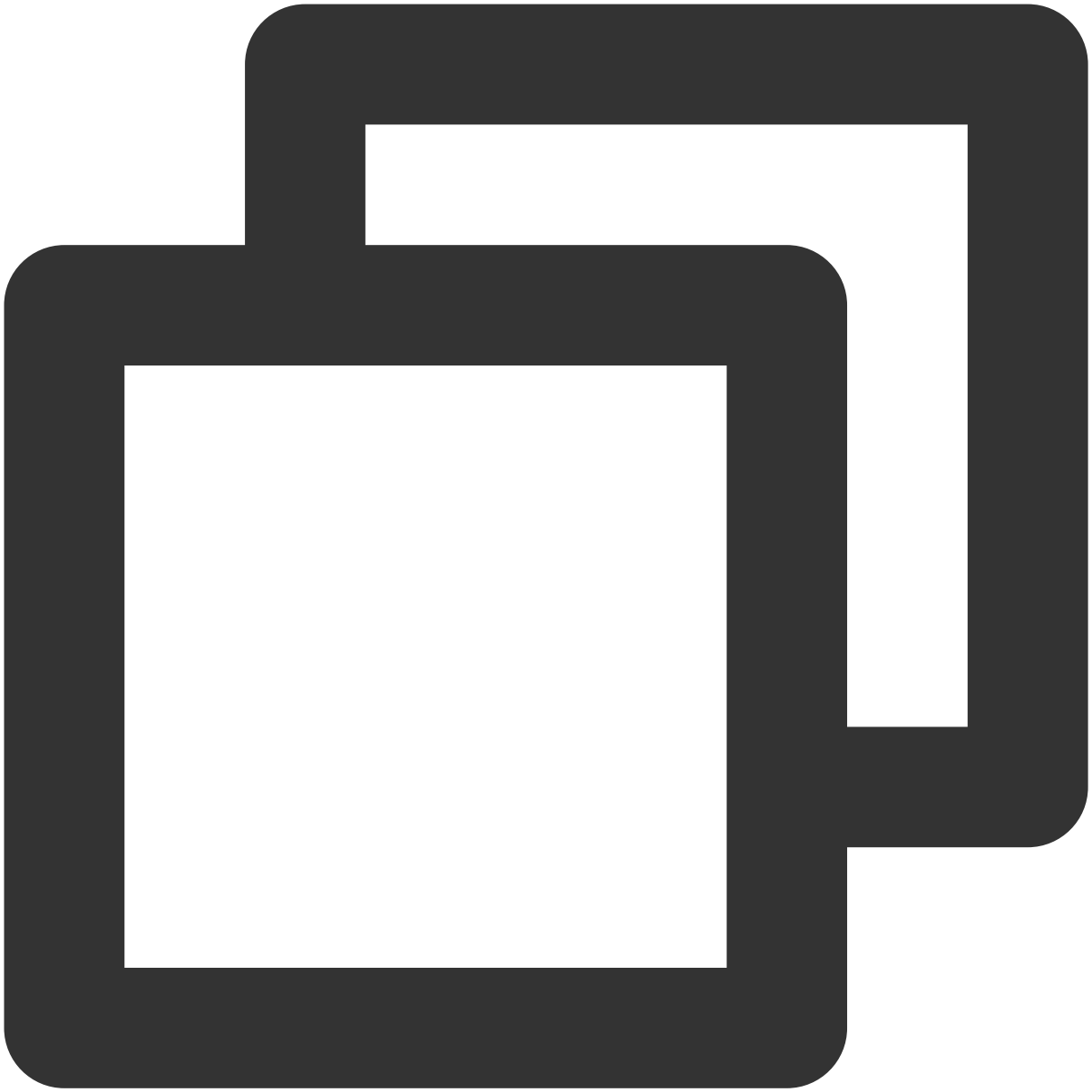


```
TUIGiftPlayView giftPlayView = new TUIGiftPlayView(mContext, roomId);
```

Note：

`TUIGiftPlayView` requires full-screen integration.

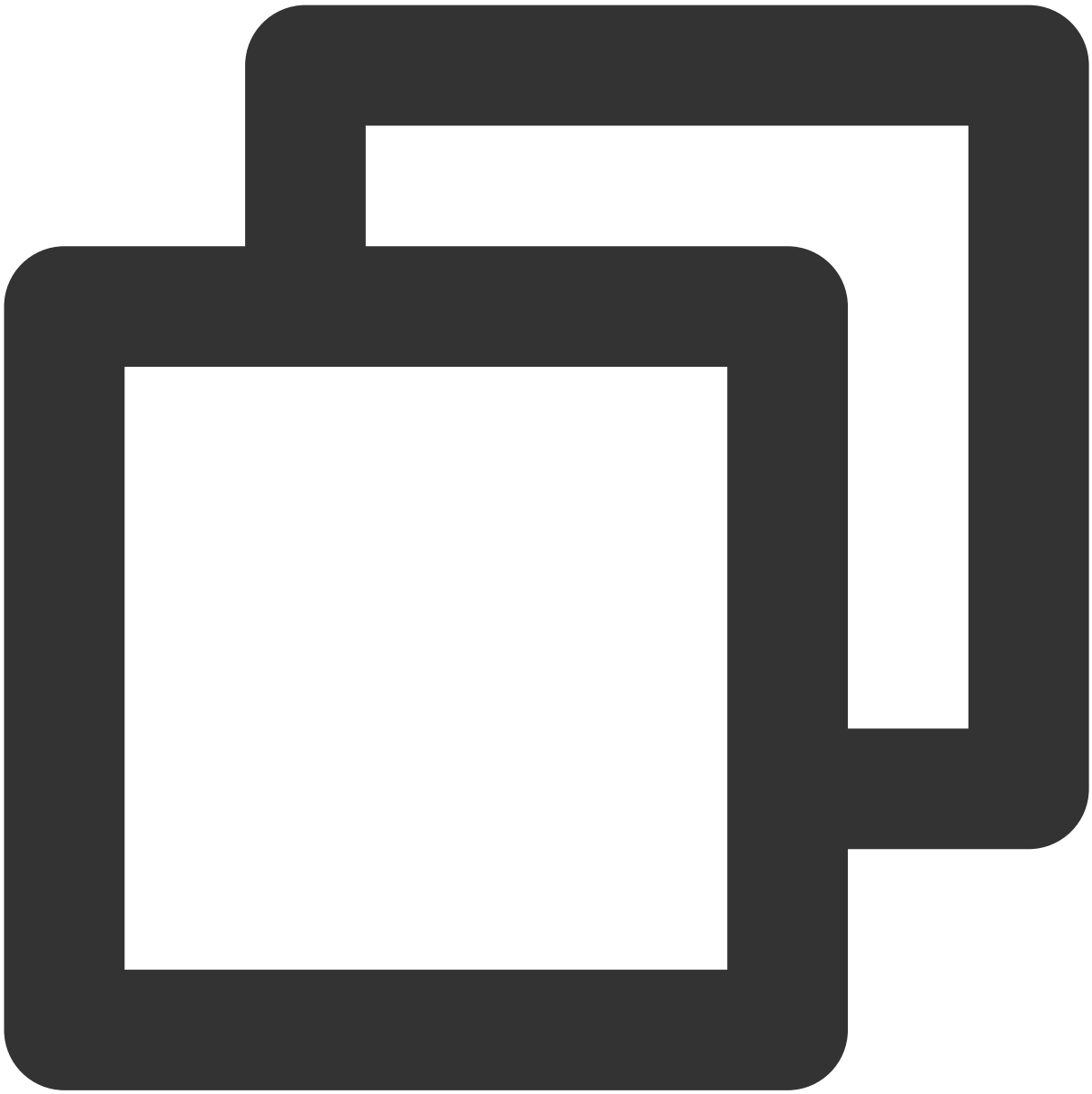
If you need to get the callback information of receiving gifts, you can implement the `onReceiveGift` callback in the `TUIGiftPlayViewListener` of `TUIGiftPlayView` .



```
public interface TUIGiftPlayViewListener {
    void onReceiveGift(TUIGift gift, int giftCount, TUIGiftUser sender, TUIGiftUser
    //...
}
```

Play Gift Animation

You need to actively invoke the `playGiftAnimation` method of `TUIGiftPlayView` when you receive `onPlayGiftAnimation` callback from the `TUIGiftPlayViewListener` of `TUIGiftPlayView` .



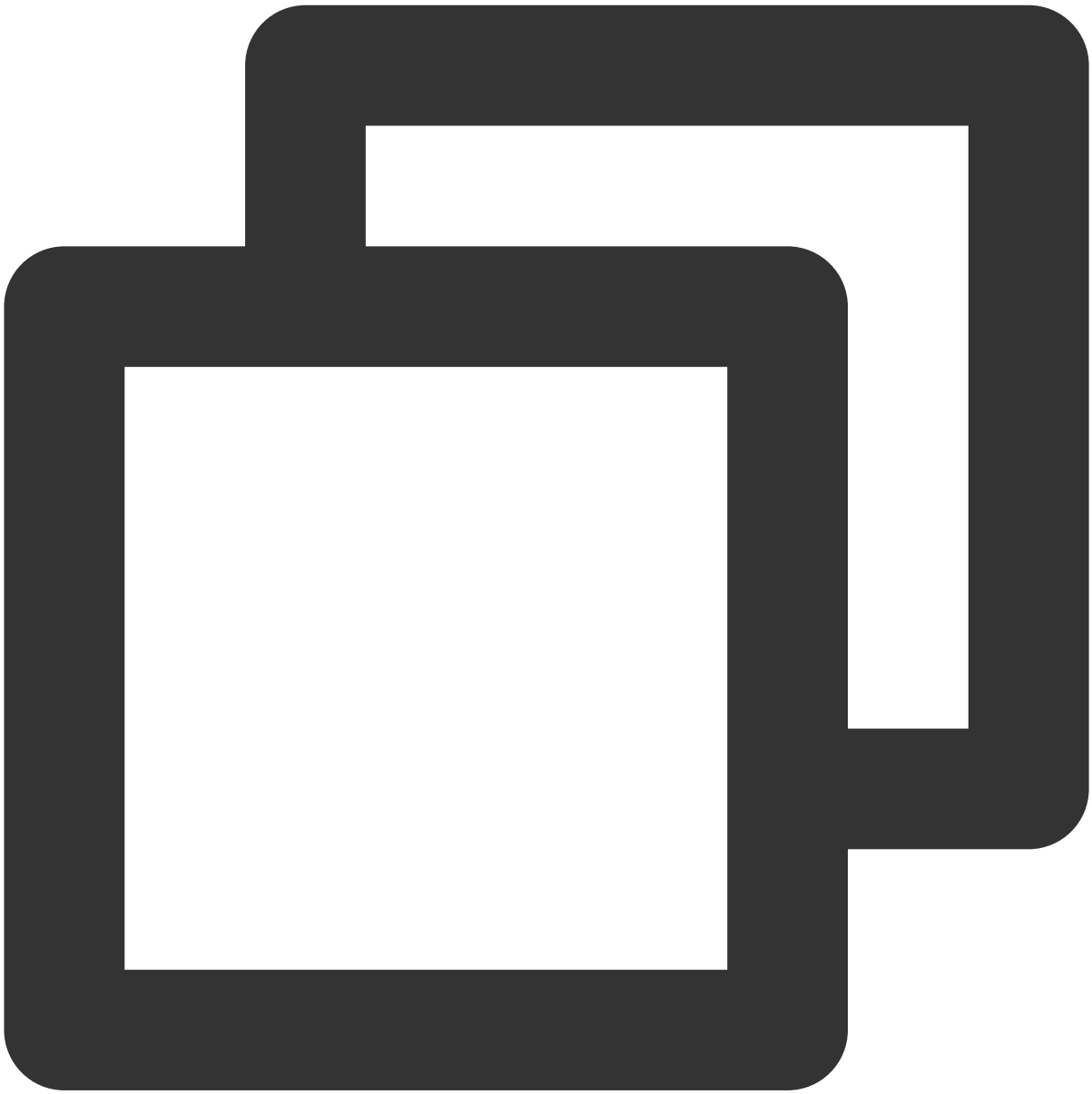
```
public interface TUIGiftPlayViewListener {
    void onPlayGiftAnimation(TUIGiftPlayView view, TUIGift gift);
    //...
}
```

Note：

Only SVGA animations are supported.

Set balance

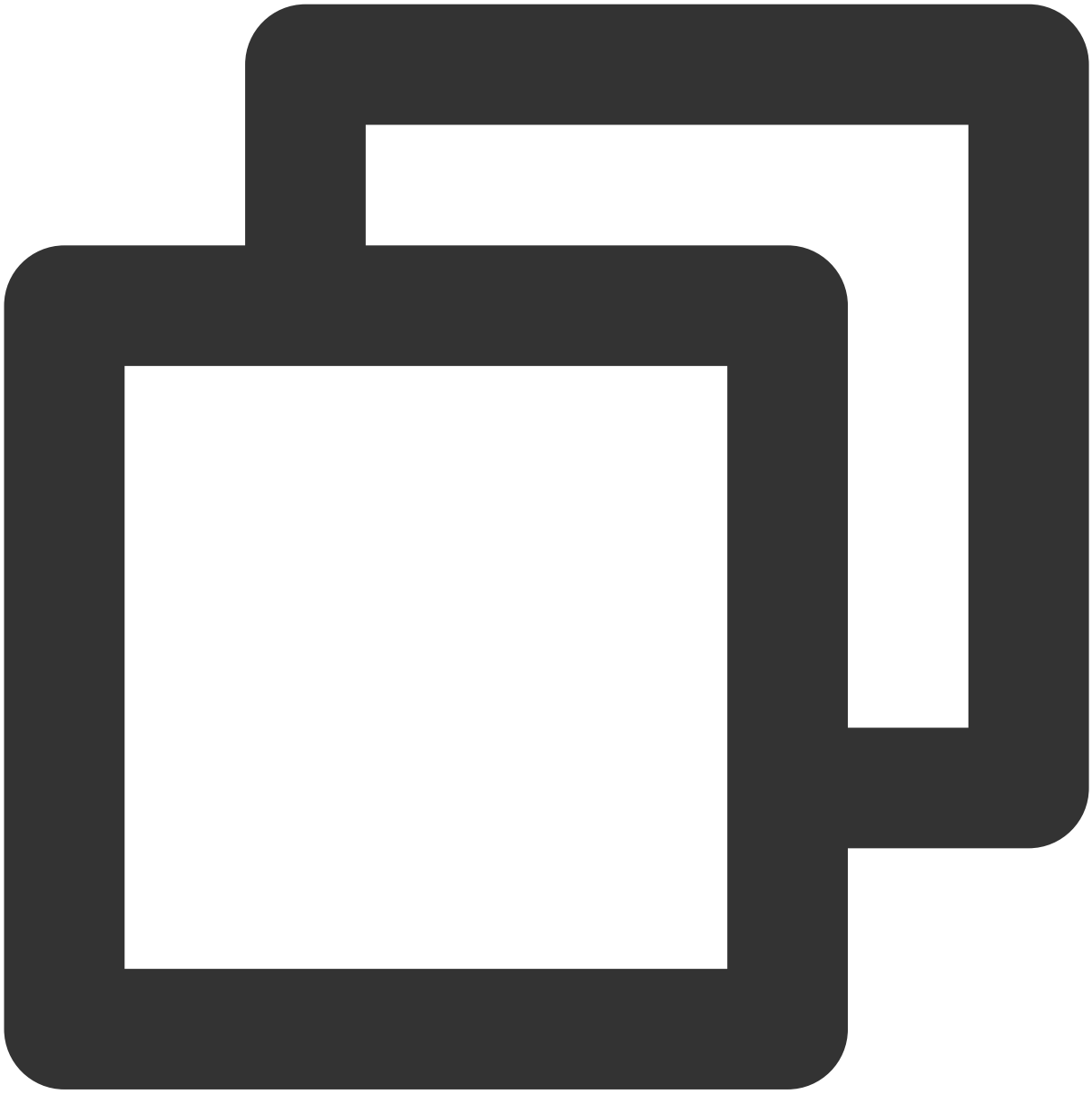
The gift panel component `TUIGiftListView` provides the `setBalance` interface, which can be used to set the balance value displayed on the gift panel.



```
giftListView.setBalance(xxx);
```

Recharge

Implementing the `onRecharge` callback in the `OnGiftListener` of `TUIGiftListView` can be used to receive the click event of the recharge button thrown by the gift display panel. Here, you can connect to your own recharge system.



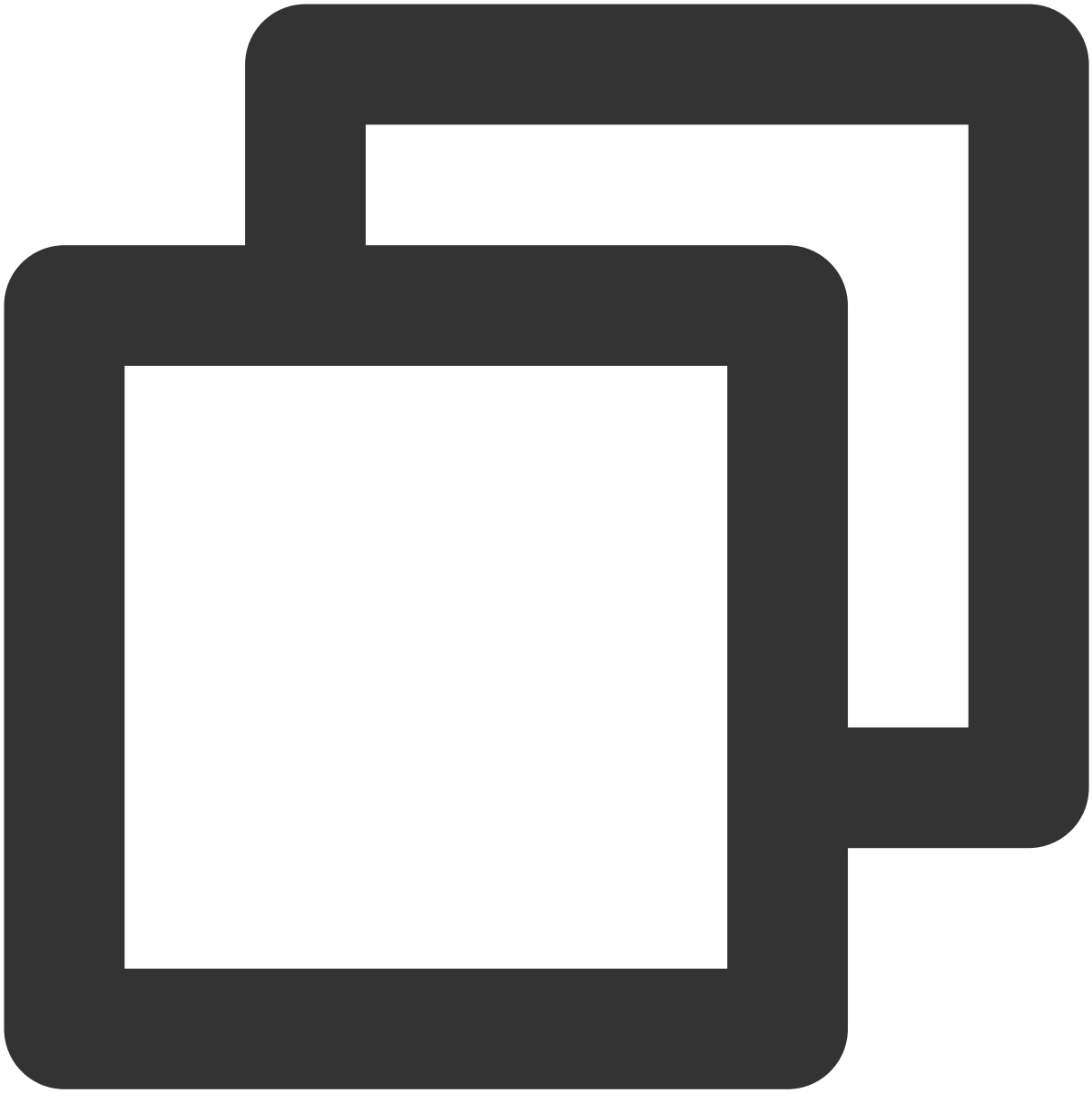
```
public void onRecharge(TUIGiftListView giftListView) {
    //...to recharge
    //setup the latest balance
    giftListView.setBalance(balance);
}
```

Note：

The gift balance is a concept of virtual currency, not real money.
The gift recharge logic is implemented externally, and customers can connect to their own recharge system. After the recharge is completed, the gift balance is updated.

Billing statistics

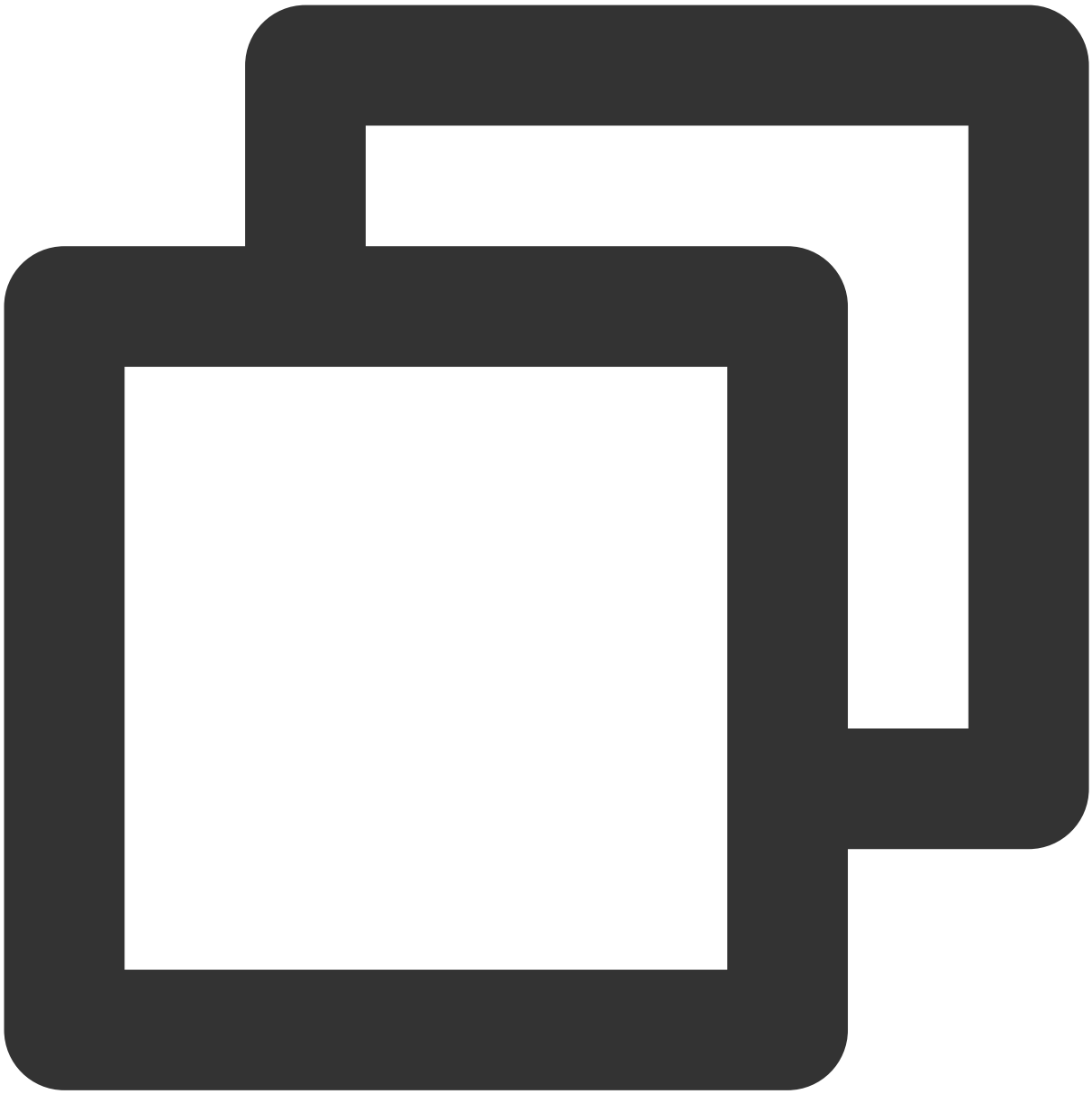
Implement the `onSendGift` callback in the `OnGiftListener` of `TUIGiftListView`, connect to the customer's own business server, complete the balance verification, gift billing, and consumption statistics, and then call the `sendGift` of `TUIGiftListView` to send the gift message.



```
public void onSendGift(TUIGiftListView giftListView, TUIGift gift, int giftCount) {
    //...Connect to the customer's own business server here to complete balance veri
    TUIGiftUser receiver = new TUIGiftUser();
    //...Set the gift receiver information here
    giftListView.sendGift(gift, giftCount, receiver);
}
```

Customize giftList

Modify the gift list on the audience's gift panel:



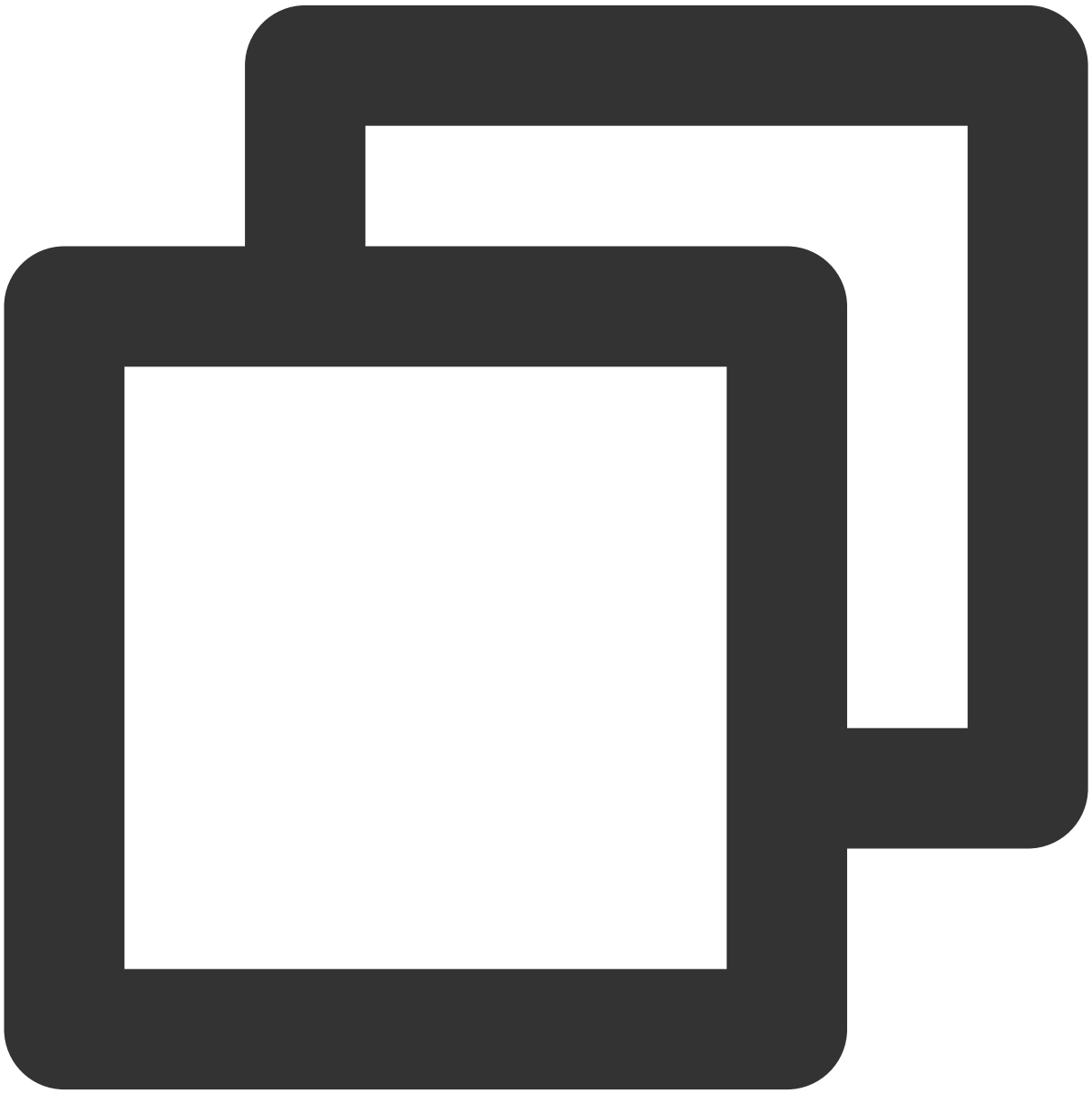
```
// Source code path：tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/a

mGiftCloudServer.queryGiftInfoList((error, result) -> post(() -> {
    if (error == Error.NO_ERROR) {
        mGiftListPanelView.setGiftList(result);
    } else {
        ToastUtil.toastLongMessage("query gift list error, code = " + error);
    }
}));
```

Note：

Customers implement the logic of `mGiftCloudServer.queryGiftInfoList` on their own, get a custom gift list `List<TUIGift>`, and set the gift list through `GiftListView.setGiftList`.

The `animationUrl` of the gift is required to be a SVGA animation.

Customize giftPanel`s balance



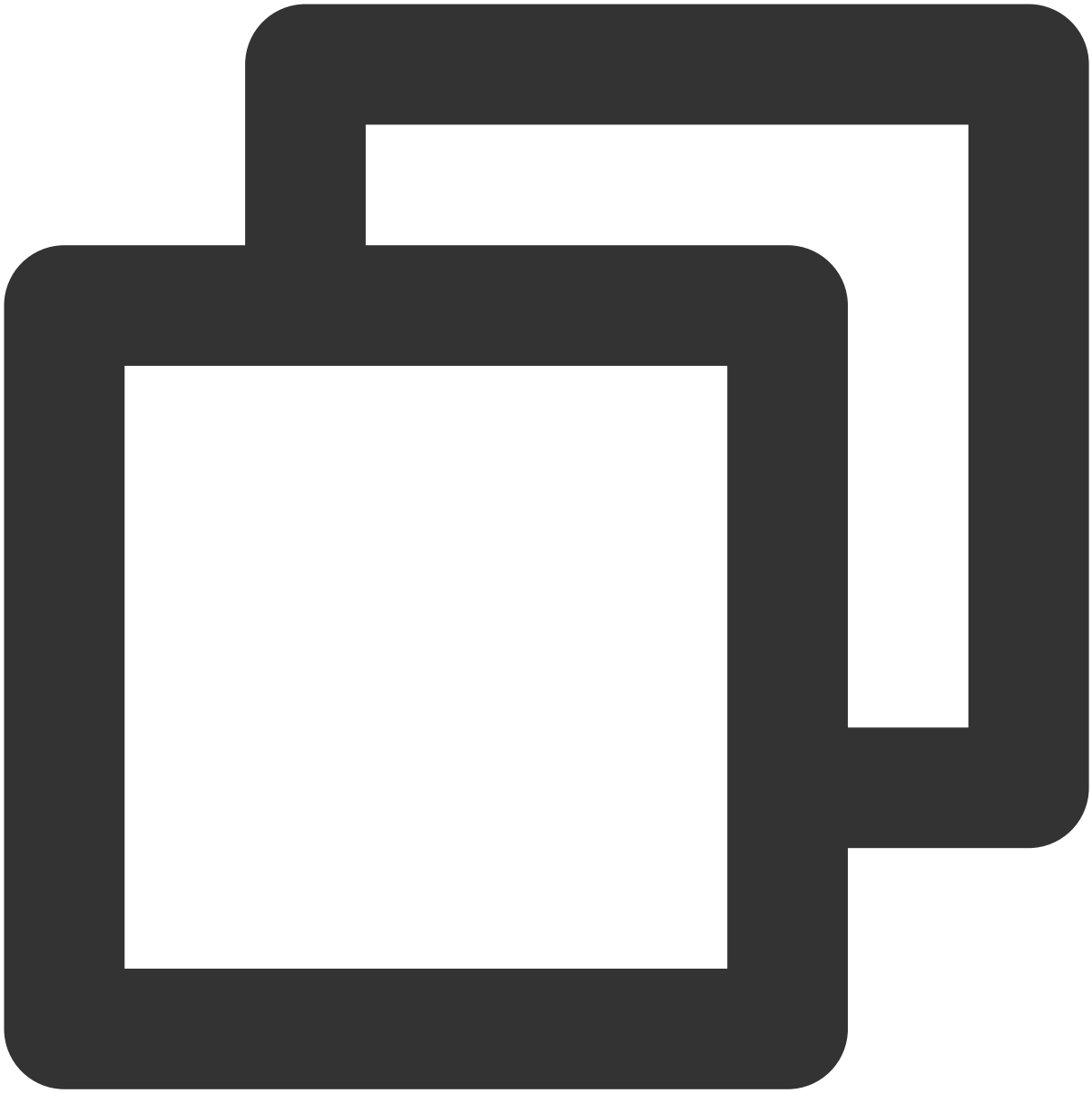
```
// Source code path：tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/a
```

```
mGiftCloudServer.queryBalance((error, result) -> post(() -> {
    if (error == Error.NO_ERROR) {
        mGiftListPanelView.setBalance(result);
    } else {
        ToastUtil.toastLongMessage("query balance error, code = " + error);
    }
}));
```

Note：

Customers implement the logic of `mGiftCloudServer.queryBalance` on their own, obtain the gift balance, and update the gift balance through `GiftListView.setBalance` .

Customize gift consumption logic



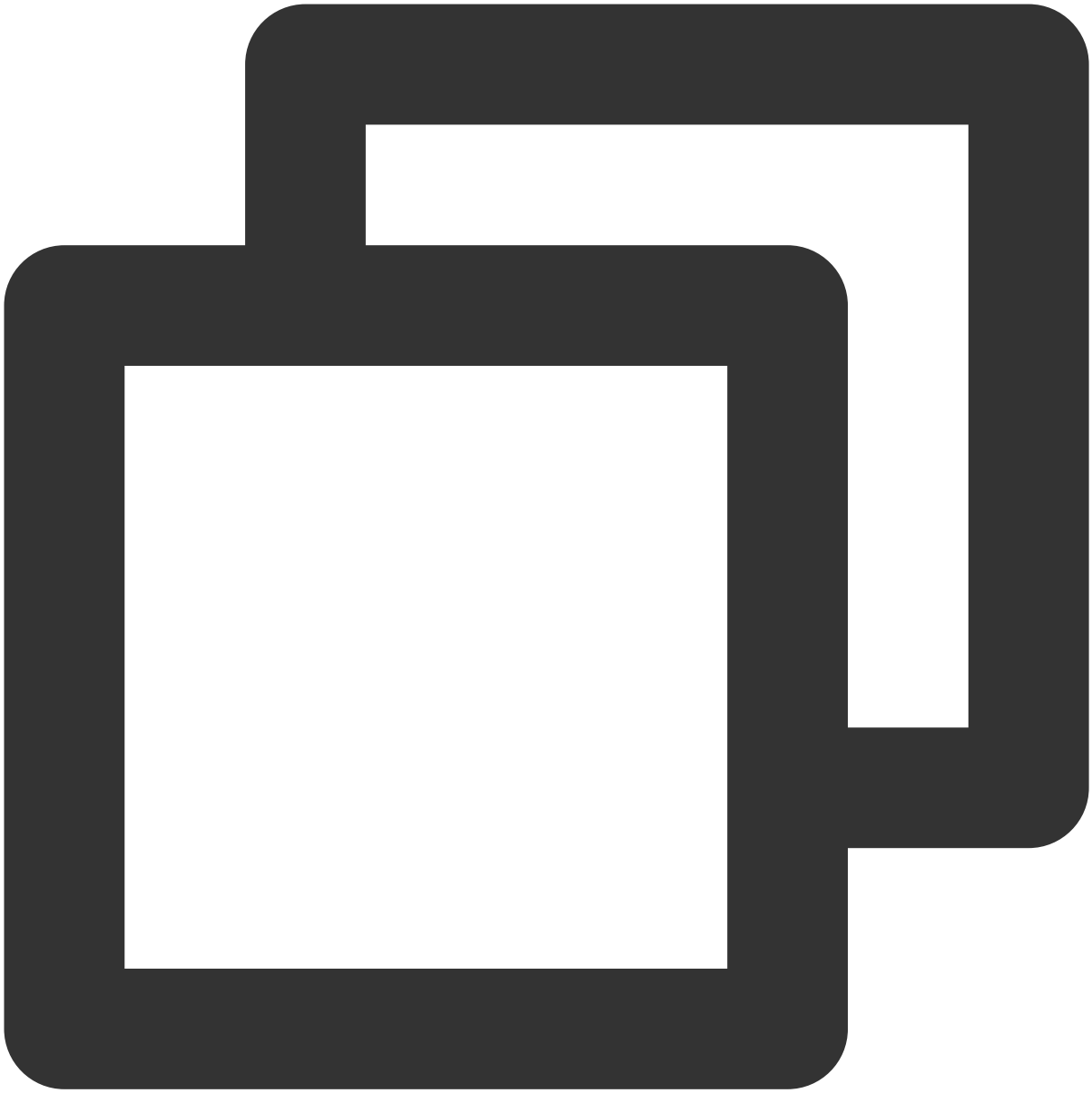
```
// Source code path：tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/a

@Override
public void onSendGift(TUIGiftListView view, TUIGift gift, int giftCount) {
    TUIGiftUser receiver = new TUIGiftUser();
    receiver.userId = mLiveRoomInfo.anchorInfo.userId;
    receiver.userName = mLiveRoomInfo.anchorInfo.name.get();
    receiver.avatarUrl = mLiveRoomInfo.anchorInfo.avatarUrl.get();
    receiver.level = "0";
    mGiftCloudServer.sendGift(TUILogin.getUserId(), receiver.userId, gift, giftCoun
            (error, result) -> post(() -> {
```

```
        if (error == Error.NO_ERROR) {
            view.sendGift(gift, giftCount, receiver);
            view.setBalance(result);
        } else {
            if (error == Error.BALANCE_INSUFFICIENT) {
                String info = getResources().getString(R.string.livekit_gift_balance
                ToastUtil.toastLongMessage(info);
            } else {
                ToastUtil.toastLongMessage("send gift error, code = " + error);
            }
        }
    }));
}
```

Note：

Customers implement the logic of `mGiftCloudServer.sendGift` on their own. The main logic is to first connect to the customer's own business server to verify the balance, and after the verification is passed, the server will charge and count the consumption records, and finally call back the result to the client. After receiving the successful callback, the client sends the gift message through the `sendGift` of the `GiftListView`, and then updates the gift balance through `setBalance`.

Customize load and play gift animation



```
// Source code path：
// tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/audience/component
// tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/anchor/component/l

@Override
public void onPlayGiftAnimation(TUIGiftPlayView view, TUIGift gift) {
    mGiftCacheService.request(gift.animationUrl, (error, result) -> {
        if (error == 0) {
            view.playGiftAnimation(result);
        }
```

```
        });
    }
```

Note：

Customers implement the logic of `mGiftCacheService.request` on their own, successfully load the animation to get the `result` (of `InputStream` type), and then play the gift animation through `playGiftAnimation` of `TUIGiftPlayView` .

Client APIs (TUICallKit)
iOS
UIKit API

Last updated：2024-07-05 19:56:15

Overview

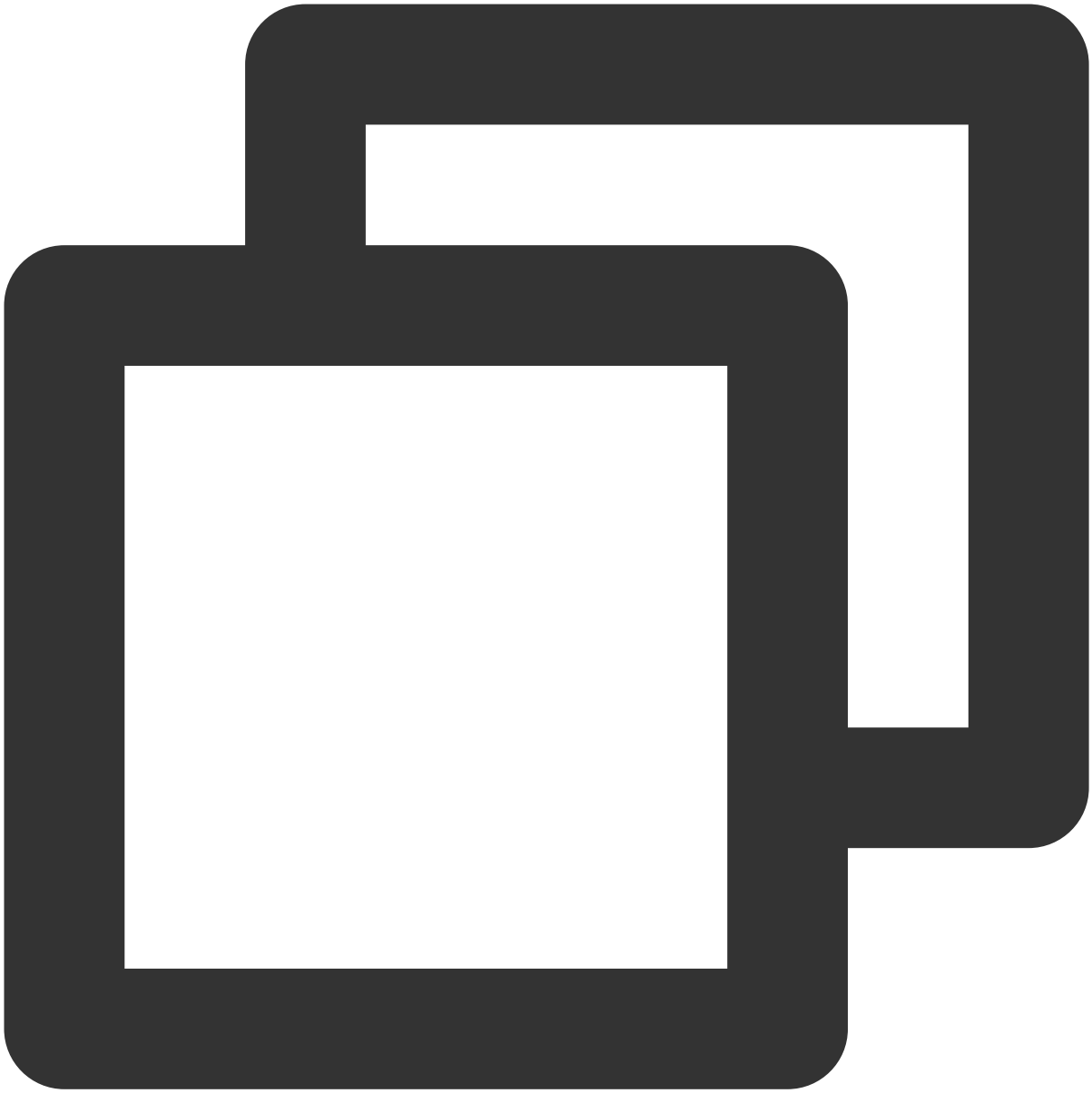
TUILiveKit is an open-source UI kit for voice chat rooms. Currently, it supports the Swift language on the iOS platform. The live streaming UI can be invoked with a simple API call.

TUIVoiceRoomViewController

API	Description
init(roomId: String, behavior: RoomBehavior, roomParams: RoomParams? = nil)	Initialize TUIVoiceRoomViewController object

init(roomId: String, behavior: RoomBehavior, roomParams: RoomParams? = nil)

Initialize TUIVoiceRoomViewController object.



```
public init(roomId: String, behavior: RoomBehavior, roomParams: RoomParams? = nil)
```

The parameters are described below:

Parameter	Type	Meaning
roomId	String	Voice Chat Room Room ID
behavior	RoomBehavior	Initialize Voice Chat Room Type

roomParams	RoomParams	The necessary parameters for creating a voice chat room, and the parameters for joining a room (with behavior as join) can be empty

RoomBehavior

Voice Chat Room Type

Type	Description
autoCreate	Directly Create a Voice Chat Room
prepareCreate	Preview Page to Create a Voice Chat Room
join	Join a Voice Chat Room

RoomParams

Voice Chat Room Parameters

Type	Description
maxSeatCount	Maximum Number of Microphones， the default value is the maximum number of seat supported by the package
seatMode	Take the Mic: Free to Join the Podium, Apply to Join the Podium (Default Application to Join the Podium)

Engine API
API Overview

Last updated：2024-07-05 19:58:20

TUIRoomEngine API List

TUIRoomEngine API is the UI-free interface of the Conference Component, which allows you to customize the encapsulation according to your business needs.
TUIRoomEngine

TUIRoomEngine Core Methods

API	Description
init	Create Instance of TUIRoomEngine.
login	Login Interface, you need to initialize user information before entering the room and perform a series of operations.
logout	Logout Interface, there will be active room leaving operation and resources destruction.
setSelfInfo	Set local user name and avatar.
getSelfInfo	Get the basic information of the local user login.
addObserver	Set Event Callback.
removeObserver	Remove Event Callback.

Active Interface related to the room

API	Description
createRoom	Create a room.
destroyRoom	Close the room.
enterRoom	Entered room.
exitRoom	Leave the room.

connectOtherRoom	Connect to another room.
disconnectOtherRoom	Disconnect from another room.
fetchRoomInfo	Get Room data.
updateRoomNameByAdmin	Update Room ID (only administrator or group owner can call).
updateRoomSpeechModeByAdmin	Set Mic Control Mode for the room (only administrator or group owner can call).

Local user view rendering and video management

API	Description
setLocalVideoView	Set the View Control for local user video rendering.
openLocalCamera	Open local Camera.
closeLocalCamera	Close local Camera.
updateVideoQuality	Update Encoding Quality settings for local video.
startPushLocalVideo	Start pushing local video.
stopPushLocalVideo	Stop pushing local video.
startScreenCapture	Start Screen Sharing (this interface is only supported on mobile devices).
startScreenCapture	Start Screen Sharing (this interface is only supported on Mac OS desktop systems).
stopScreenCapture	End Screen Sharing.
getScreenCaptureSources	Enumerate shareable screens and windows (this interface is only supported on Mac OS systems).
selectScreenCaptureTarget	Select the screen or window to share (this interface is only supported on Mac OS systems).

Local user audio management

API	Description
openLocalMicrophone	Open local mic.
closeLocalMicrophone	Close local mic.

updateAudioQuality	Update local Audio Encoding Quality settings.
startPushLocalAudio	Start pushing local audio.
stopPushLocalAudio	Stop pushing local audio.

Remote user view rendering and video management

API	Description
setRemoteVideoView	Set the View Control for remote user video rendering.
startPlayRemoteVideo	Start Playback of remote user video.
stopPlayRemoteVideo	Stop Playback of remote user video.
muteRemoteAudioStream	Mute remote user.

Room user information

API	Description
getUserList	Get the member list in the room.
getUserInfo	Get member information.

Room user management

API	Description
changeUserRole	Modify user role (only administrator or group owner can call).
kickRemoteUserOutOfRoom	Kick remote user out of the room (only administrator or group owner can call).

Room user speech management

API	Description
disableDeviceForAllUserByAdmin	Control the permission status of all users in the current room to open audio streams, video streams, and capture devices, such as: all users are prohibited from opening mics, all users are prohibited from opening cameras, all users are prohibited from opening screen sharing (currently only available in conference scenarios, and only administrators or group owners can call).

openRemoteDeviceByAdmin	Request remote user to open media device (only administrator or group owner can call).
closeRemoteDeviceByAdmin	Close remote user media device (only administrator or group owner can call).
applyToAdminToOpenLocalDevice	Request to open local media device (available for ordinary users).

Room mic seat management

API	Description
setMaxSeatCount	Set the maximum number of mic seats (only supported when entering the room and creating the room).
getSeatList	Get the list of mic seats.
lockSeatByAdmin	Lock the mic seat (only administrator or group owner can call, including position lock, audio status lock, and video status lock).
takeSeat	Apply to Go Live (no need to apply in free speech mode).
leaveSeat	Apply to leave the mic (no need to apply in free speech mode).
takeUserOnSeatByAdmin	Host/Administrator invites user to Go Live.
kickUserOffSeatByAdmin	Host/Administrator kicks user off the mic.

Signaling management

API	Description
cancelRequest	Cancel Request.
responseRemoteRequest	Reply Request.

Send message

API	Description
sendTextMessage	Send Text Message.

sendCustomMessage	Send Custom Message.
disableSendingMessageByAdmin	Disable remote user's ability to send text messages (only administrator or group owner can call).
disableSendingMessageForAllUser	Disable all users' ability to send text messages (only administrator or group owner can call).

Advanced features: Get TRTC instance

API	Description
getDeviceManager	Get native TRTC Device Management class.
getAudioEffectManager	Get native TRTC Sound Effect Class.
getBeautyManager	Get native TRTC Beauty Class.
getTRTCCloud	Get native TRTC Instance Class.

TUIRoomObserver Callback Events

TUIRoomObserver is the callback event class corresponding to TUIRoomEngine. You can listen to the callback events you need through this callback.
TUIRoomObserver

TUIRoomObserver

Error callback

API	Description
onError	Error Event Callback.

Login status event callback

API	Description
onKickedOffLine	Other terminal login is kicked offline.

onUserSigExpired	User Credential Timeout Event.

Room event callback

API	Description
onRoomNameChanged	Room ID change event.
onAllUserMicrophoneDisableChanged	All users in the room have their mics disabled event.
onAllUserCameraDisableChanged	All users in the room have their cameras disabled event.
onSendMessageForAllUserDisableChanged	All users in the room have their text message sending disabled event.
onKickedOutOfRoom	Kicked out of the room event.
onRoomDismissed	Room closed event.
onRoomSpeechModeChanged	Room Mic Control Mode changed.

Room user event callback

API	Description
onRemoteUserEnterRoom	Remote user entered room event.
onRemoteUserLeaveRoom	Remote user left the room event.
onUserRoleChanged	User role changed event.
onUserVideoStateChanged	User video status changed event.
onUserAudioStateChanged	User audio status changed event.
onUserScreenCaptureStopped	User screen capture stopped event.
onUserVoiceVolumeChanged	User volume change event.
onSendMessageForUserDisableChanged	User text message sending ability changed event.
onUserNetworkQualityChanged	User network status change event.

Room mic seat event callback

API	Description

onRoomMaxSeatCountChanged	Maximum mic seat changed event in the room (only effective in conference type rooms).
onSeatListChanged	Mic seat list changed event.
onKickedOffSeat	Received user kicked off mic event.

Request signaling event callback

API	Description
onRequestReceived	Received request message event.
onRequestCancelled	Received request cancelled event.

Room message event callback

API	Description
onReceiveTextMessage	Received normal text message event.
onReceiveCustomMessage	Received custom message event.

Android
UIKit API

Last updated：2024-07-05 19:56:37

Overview

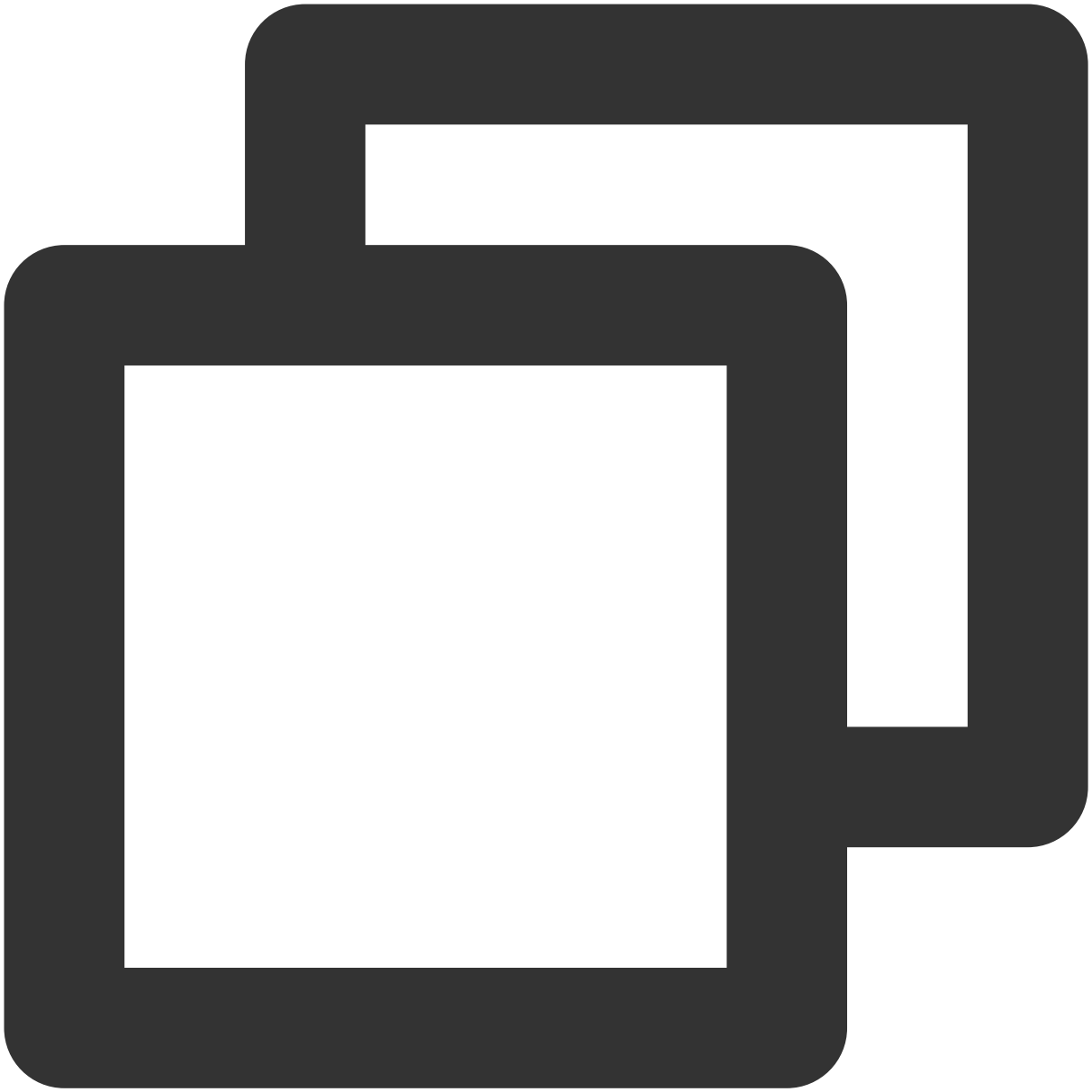
TUILiveKit is a UI open-source kit for voice chat rooms. Currently, the Android platform only supports the Java language. The live streaming UI can be invoked with a simple API call.

TUIVoiceRoomFragment Class

API	Description
TUIVoiceRoomFragment(String roomId, LiveDefine.RoomBehavior behavior, LiveDefine.RoomParams params)	Construct Voice Chat Room Main Interface objects

TUIVoiceRoomFragment

Construct Direct Interface objects



```
public TUIVoiceRoomFragment(String roomId, LiveDefine.RoomBehavior behavior, LiveDe
```

The parameters are described below:

Parameter	Type	Meaning
roomId	String	Voice Chat Room Room ID
behavior	RoomBehavior	Initialize Voice Chat Room Type
params	RoomParams	The necessary parameters for creating a voice chat room,

		and the parameters for joining a room (with behavior as JOIN) can be empty.

RoomBehavior

Voice Chat Room Type

Type	Description
AUTO_CREATE	Directly Create a Voice Chat Room
PREPARE_CREATE	Preview Page to Create a Voice Chat Room
JOIN	Join a Voice Chat Room

RoomParams

Voice Chat Room Parameters

Type	Description
maxSeatCount	Maximum Number of Microphones， the default value is the maximum number of seat supported by the package
seatMode	Take the Mic: Free to Join the Podium, Apply to Join the Podium (Default Application to Join the Podium)

Engine API
API Overview

Last updated：2024-07-05 19:58:32

TUIRoomEngine (No UI Interface)

TUIRoomEngine API is the Audio/Video call Component's No UI Interface, you can use this set of API to customize packaging according to your business needs.
TUIRoomEngine

TUIRoomEngine Core Methods

API	Description
createInstance	Create TUIRoomEngine Instance
destroyInstance	Destroy TUIRoomEngine Instance
login	Login interface, you need to initialize user information before entering the room and perform a series of operations.
logout	Logout interface, there will be actively leave room operation, destroy resources
setSelfInfo	Set local user name and avatar
getSelfInfo	Get local user basic information
addObserver	Set event callback
removeObserver	Remove event callback

Room Related Active Interface

API	Description
createRoom	Create room
destroyRoom	Close the room
enterRoom	Entered room
exitRoom	Leave room

connectOtherRoom	Connect to other room
disconnectOtherRoom	Disconnect from other room
fetchRoomInfo	Get room data
updateRoomNameByAdmin	Update room name
updateRoomSpeechModeByAdmin	Set room management mode (only administrator or group owner can call)

Local User View Rendering, Video Management

API	Description
setLocalVideoView	Set the view control for local user video rendering
openLocalCamera	Open local camera
closeLocalCamera	Close local camera
updateVideoQuality	Update local video codec quality settings
startScreenSharing	Start screen sharing
stopScreenSharing	End screen sharing
startPushLocalVideo	Start pushing local video
stopPushLocalVideo	Stop pushing local video

Local User Audio Management

API	Description
openLocalMicrophone	Open local microphone
closeLocalMicrophone	Close local microphone
updateAudioQuality	Update local audio codec quality settings
startPushLocalAudio	Start pushing local audio
stopPushLocalAudio	Stop pushing local audio

Remote User View Rendering, Video Management

API	Description

setRemoteVideoView	Set the view control for remote user video rendering
startPlayRemoteVideo	Start playing remote user video
stopPlayRemoteVideo	Stop playing remote user video
muteRemoteAudioStream	Mute remote user

Room User Information

API	Description
getUserList	Get the member list in the room
getUserInfo	Get member information

Room User Management

API	Description
changeUserRole	Modify user role (only administrator or group owner can call)
kickRemoteUserOutOfRoom	Kick Remote User out of the Room (Only Administrator or Group Owner can call)

Speech Management in Room

API	Description
disableDeviceForAllUserByAdmin	Media Device Management for All Users (Only Administrator or Group Owner can call)
openRemoteDeviceByAdmin	Request Remote User to Open Media Device (Only Administrator or Group Owner can call)
closeRemoteDeviceByAdmin	Close Remote User's Media Device (Only Administrator or Group Owner can call)
applyToAdminToOpenLocalDevice	Request to Open Local Media Device (Available for Ordinary Users)

Microphone Seat Management in Room

API	Description
setMaxSeatCount	Set Maximum Number of Microphone Seats (Only supported when entering the

	room and creating the room)
getSeatList	Get Microphone Seat List
lockSeatByAdmin	Lock Microphone Seat (Including Position Lock, Audio State Lock, Video State Lock)
takeSeat	Go Live Locally Conference Scene: SPEAK_AFTER_TAKING_SEAT mode requires application to the host or administrator to allow going live, other modes do not support going live. Live Broadcast Scene: FREE_TO_SPEAK mode allows free going live, and speak after going live; SPEAK_AFTER_TAKING_SEAT mode requires application to the host or administrator to allow going live; other modes do not support going live.
leaveSeat	Leave Microphone Seat Locally
takeUserOnSeatByAdmin	Host/Administrator invites user to go live
kickUserOffSeatByAdmin	Host/Administrator kicks user off the microphone seat

Signaling Management

API	Description
cancelRequest	Cancel Request
responseRemoteRequest	Reply to Request

Send Message

API	Description
sendTextMessage	Send Text Message
sendCustomMessage	Send Custom Message
disableSendingMessageByAdmin	Disable Remote User's Text Message Sending Ability (Only Administrator or Group Owner can call)
disableSendingMessageForAllUser	Disable All Users' Text Message Sending Ability (Only Administrator or Group Owner can call) Advanced Feature: Get TRTC Instance

Advanced Feature: Get TRTC Instance

API	Description

getTRTCCloud	Get TRTC Instance Object
getDeviceManager	Get Device Management Object
getAudioEffectManager	Get Audio Effect Management Object
getBeautyManager	Get Beauty Management Object

Event Type Definition

TUIRoomObserver is the Callback Event class corresponding to TUIRoomEngine. You can listen to the callback events you need through this callback.
TUIRoomObserver

Error Callback

Event	Description
onError	Error Callback Event

Login Status Event Callback

API	Description
onKickedOffLine	User Kicked Offline Event
onUserSigExpired	User Credential Timeout Event

Room Event Callback

API	Description
onRoomNameChanged	Room Name Change Event
onAllUserMicrophoneDisableChanged	All Users' Microphones Disabled in Room Event
onAllUserCameraDisableChanged	All Users' Cameras Disabled in Room Event
onSendMessageForAllUserDisableChanged	All Users' Text Message Sending Disabled in Room Event
onRoomDismissed	Room Dismissed Event

onKickedOutOfRoom	Kicked Out of Room Event
onRoomSpeechModeChanged	Room Microphone Control Mode Change

Room User Event Callback

API	Description
onRemoteUserEnterRoom	Remote User Entering Room Event
onRemoteUserLeaveRoom	Remote User Leaving Room Event
onUserRoleChanged	User Role Change Event
onUserVideoStateChanged	User Video State Change Event
onUserAudioStateChanged	User Audio State Change Event
onUserVoiceVolumeChanged	User Volume Change Event
onSendMessageForUserDisableChanged	User Text Message Sending Ability Change Event
onUserNetworkQualityChanged	User Network Status Change Event
onUserScreenCaptureStopped	Screen Sharing End Event

Room Microphone Seat Event Callback

API	Description
onRoomMaxSeatCountChanged	Room Maximum Microphone Seat Number Change Event (Only effective in conference type rooms)
onSeatListChanged	Microphone Seat List Change Event
onKickedOffSeat	Received User Kicked Off Microphone Event

Request Signaling Event Callback

API	Description
onRequestReceived	Received Request Message Event
onRequestCancelled	Received Request Cancellation Event

Room Message Event Callback

API	Description
onReceiveTextMessage	Received Normal Text Message Event
onReceiveCustomMessage	Received Custom Message Event

Error Codes (TUILiveKit)

Last updated：2024-05-17 11:20:40

General Error Code

Error Code	Description
0	Operation Successful
-1	Temporarily Unclassified General Error
-2	Request Rate Limited, Please Try Again Later
-1000	Not Found SDKAppID, Please Confirm Application Info in TRTC Console
-1001	Passing illegal parameters when calling API, check if the parameters are legal
-1002	Not Logged In, Please Call Login API
-1003	Failed to Obtain Permission, Unauthorized Audio/Video Permission, Please Check if Device Permission is Enabled
-1004	This feature requires an additional package. Please activate the corresponding package as needed in the TRTC Console

Local User Rendering, Video Management, Audio Management API Callback Error Definition

Error Code	Description
-1100	System Issue, Failed to Open Camera. Check if Camera Device is Normal
-1101	Camera has No System Authorization, Check System Authorization
-1102	Camera is Occupied, Check if Other Process is Using Camera
-1103	No Camera Device Currently, Please Insert Camera Device to Solve the Problem
-1104	System Issue, Failed to Open Mic. Check if Mic Device is Normal
-1105	Mic has No System Authorization, Check System Authorization
-1106	Mic is Occupied

-1107	No Mic Device Currently
-1108	Failed to Obtain Screen Sharing Object, Check Screen Recording Permission
-1109	Failed to Enable Screen Sharing, Check if Someone is Already Screen Sharing in the Room

Room Management Related API Callback Error Definition

Error Code	Description
-2100	Room Does Not Exist When Entering, May Have Been Closed
-2101	This Feature Can Only Be Used After Entering the Room
-2102	Room Owner Does Not Support Leaving the Room, Conference Room Type: Transfer Room Ownership First, Then Leave the Room. Living Room Type: Room Owner Can Only Close the Room
-2103	This Operation is Not Supported in the Current Room Type
-2104	This Operation is Not Supported in the Current Speaking Mode
-2105	Illegal Custom Room ID, Must Be Printable ASCII Characters (0x20-0x7e), Up to 48 Bytes Long
-2106	Room ID is Already in Use, Please Choose Another Room ID
-2107	Illegal Room Name, Maximum 30 Bytes, Must Be UTF-8 Encoding if Contains Chinese Characters
-2108	User is Already in Another Room, Single RoomEngine Instance Only Supports User Entering One Room, To Enter Different Room, Please Leave the Room or Use New RoomEngine Instance

Room User Information API Callback Error Definition

Error Code	Description
-2200	User Not Found
-2201	User Not Found in the Room

Room User Speech Management API Callback Error Definition & Room Mic Seat Management API Callback Error Definition

Error Code	Description

-2300	Room Owner Permission Required for Operation
-2301	Room Owner or Administrator Permission Required for Operation
-2310	No Permission for Signaling Request, e.g. Canceling an Invite Not Initiated by Yourself
-2311	Signaling Request ID is Invalid or Has Been Processed
-2340	Maximum Mic Seat Exceeds Package Quantity Limit
-2341	Current User is Already on Mic Seat
-2342	Mic Seat is Already Occupied
-2343	Mic Seat is Locked
-2344	Mic Seat Serial Number Does Not Exist
-2345	Current User is Not on Mic
-2346	Mic-on Capacity is Full
-2360	Current Mic Seat Audio is Locked
-2361	Need to Apply to Room Owner or Administrator to Open Mic
-2370	Current Mic Seat Video is Locked, Need Room Owner to Unlock Mic Seat Before Opening Camera
-2371	Need to Apply to Room Owner or Administrator to Open Camera
-2380	All Members Muted in the Current Room
-2381	You Have Been Muted in the Current Room

Release Notes（TUILiveKit）
iOS

Last updated：2024-05-17 11:20:40

May 2024

Post updates	Description	Release Date
Version 1.0.0	Support custom Definition gifts Support custom Definition bullet comments Support sound effects & reverb Support likes Support seat controller	2024.05.17

Android

Last updated：2024-05-17 11:20:40

May 2024

Post updates	Describe	Release time
Version 1.0.0	Support custom Definition gifts Support custom Definition bullet comments Support sound effects & reverb Support likes Support seat controller	2024.05.17

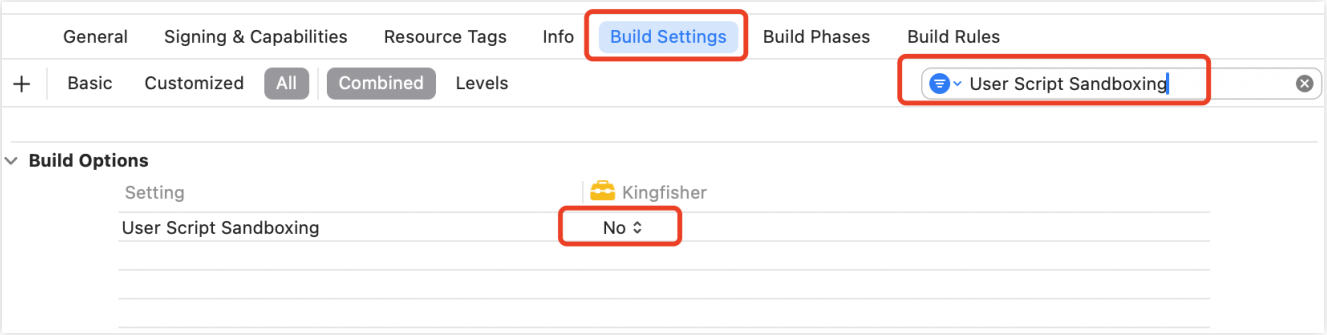
FAQs（TUILiveKit）
iOS

Last updated：2024-05-17 11:48:33

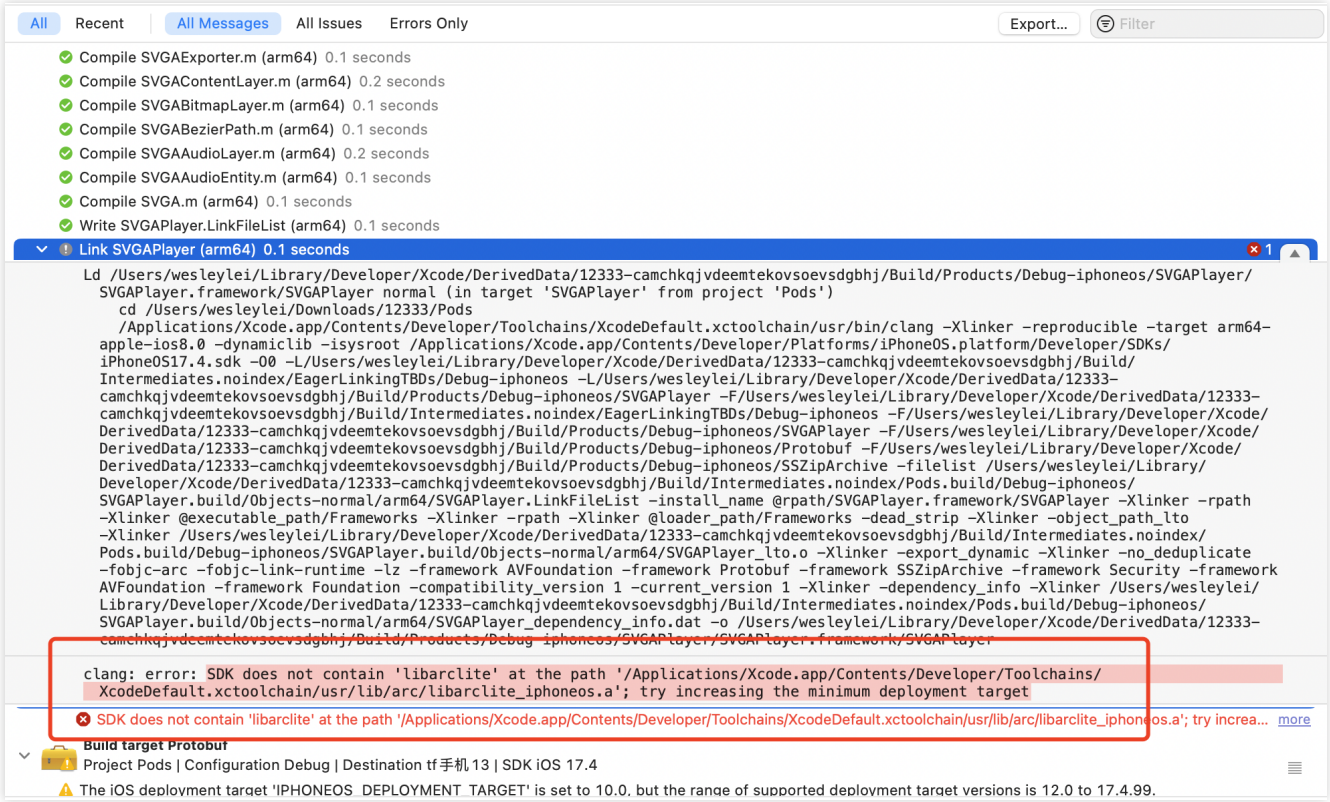
Xcode 15 compiler error?

1. **Sandbox: rsync** is displayed.

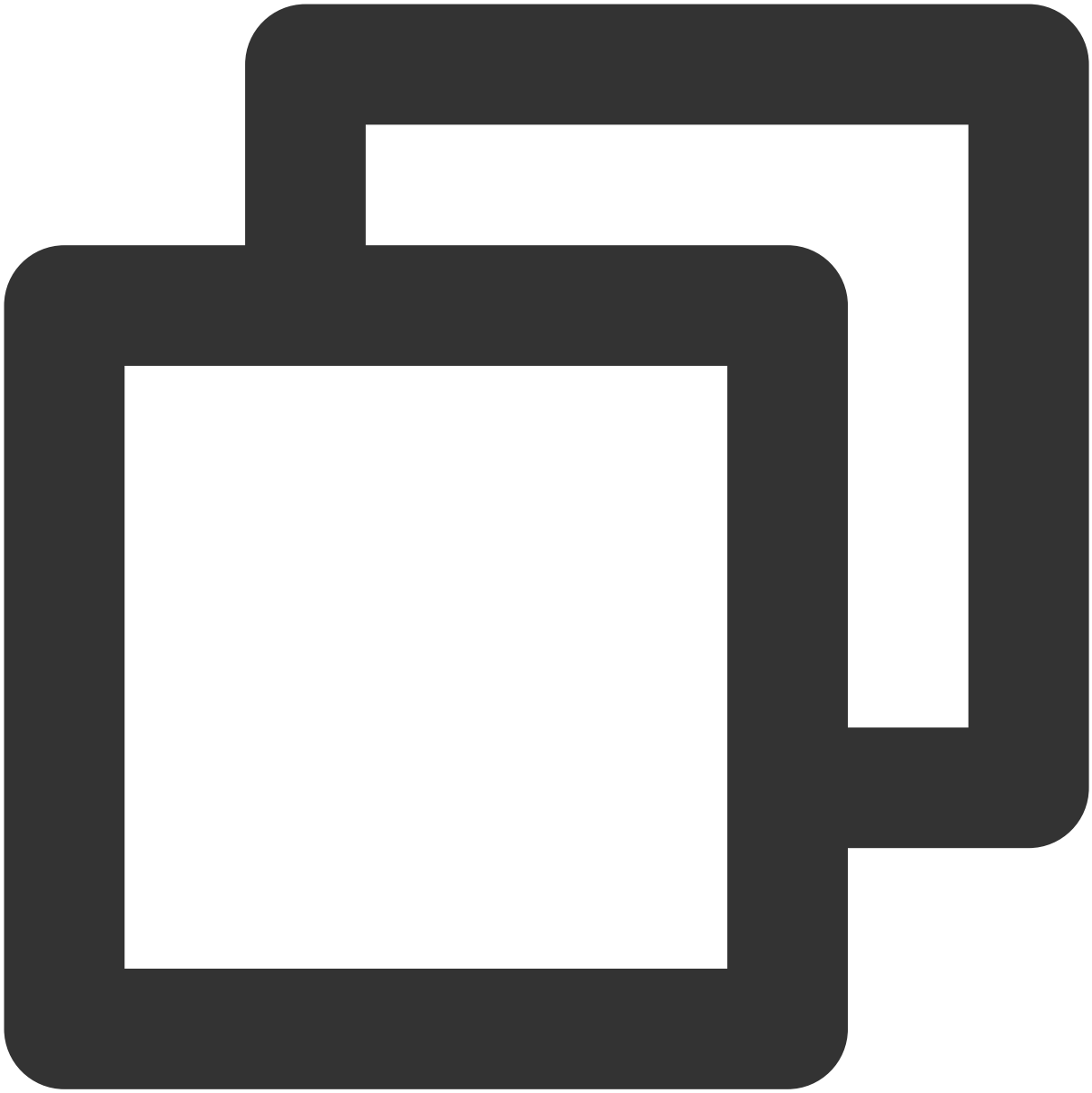
You can set User Script Sandboxing to **NO** in **Build Settings:**



2. If **SDK does not contain**, compile error screenshot:

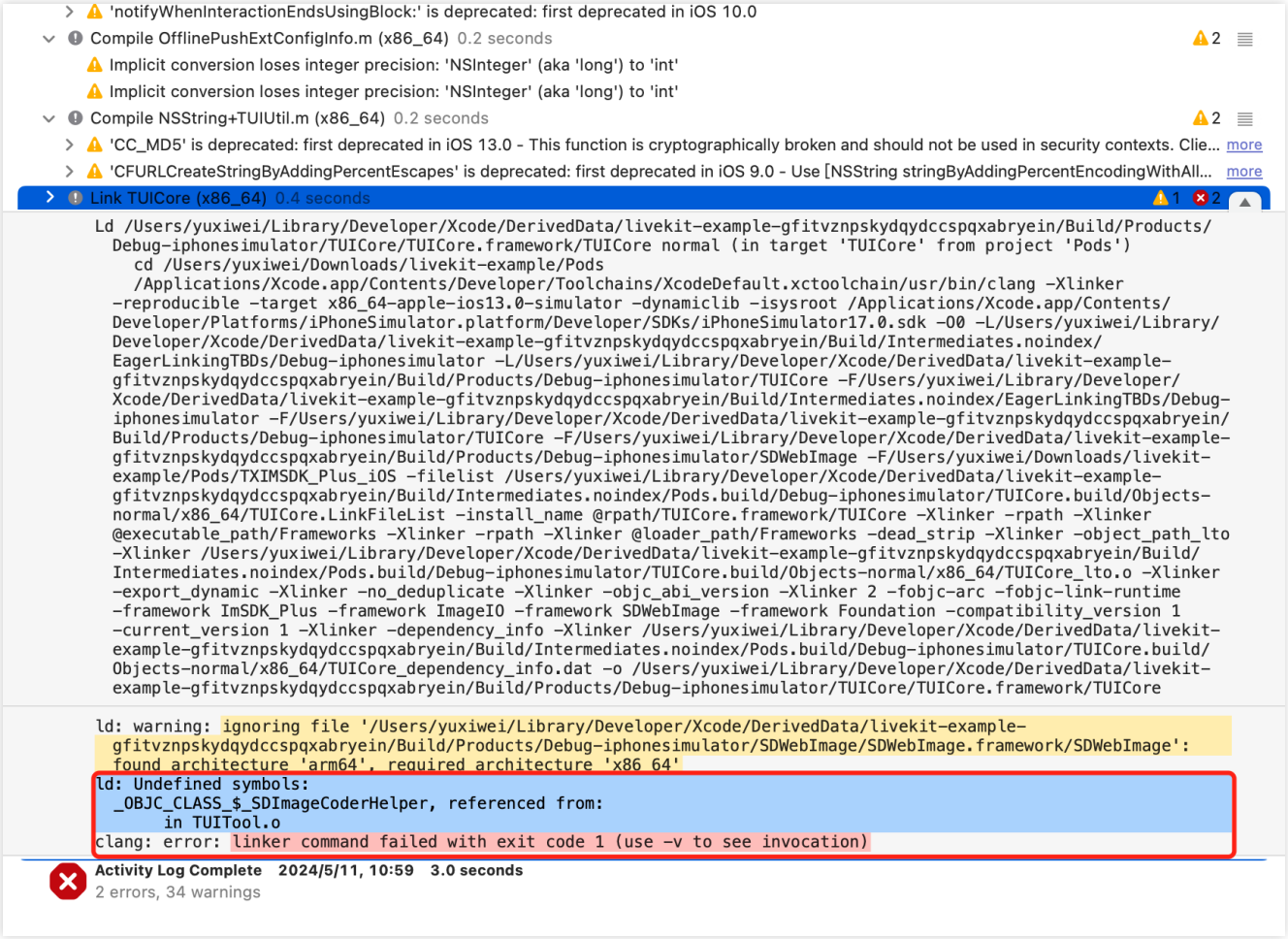


Add the following code to the Podfile:

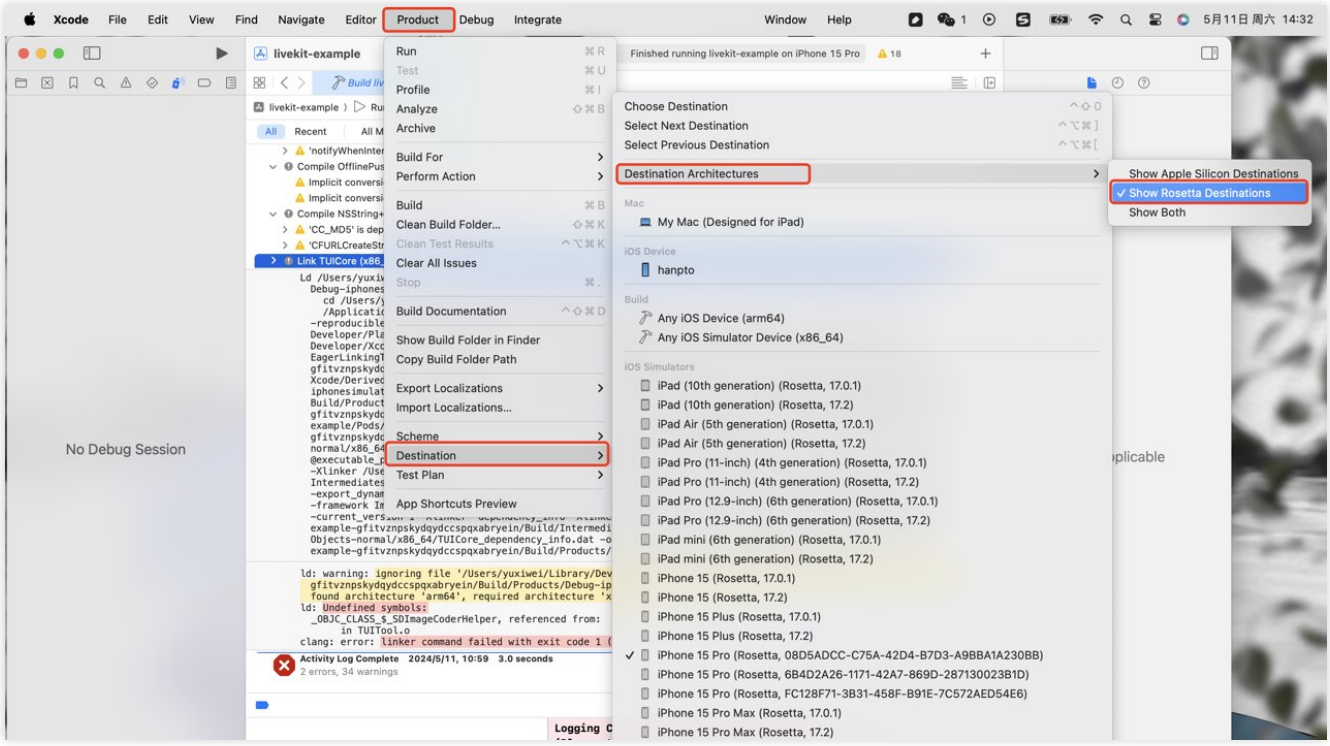


```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '13.0'
    end
  end
end
```

3. If you run the emulator on an M-series computer, **Linker command failed with exit code 1 (use-v to see invocation)** may appear.



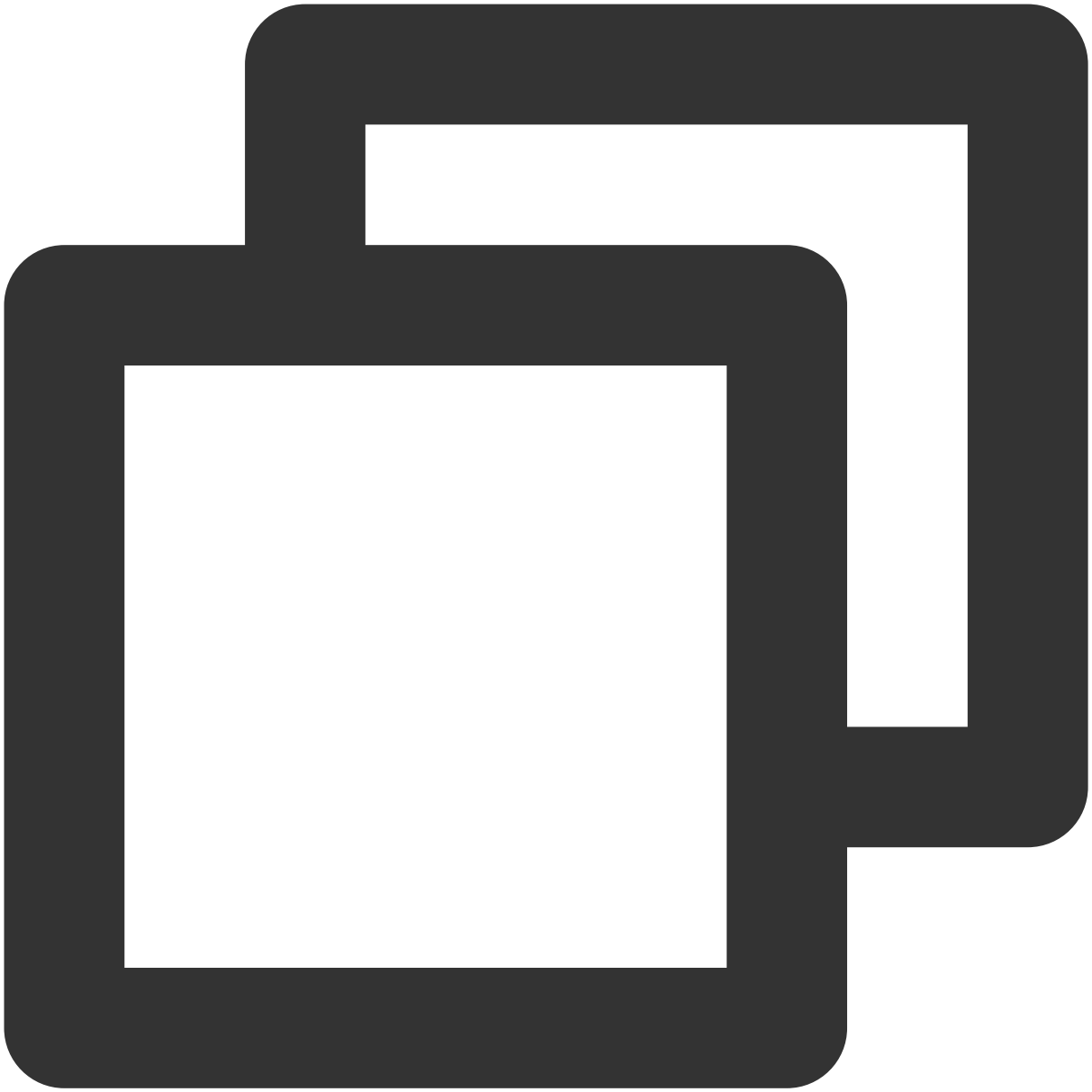
The xcode configuration needs to be modified. xcode open projects > **Product > Destination > Destination Architectures** can choose which mode of emulator to open with, and need to select the ending emulator (**Rosetta**).



Is there a conflict between TUILiveKit and the integrated audio and video library?

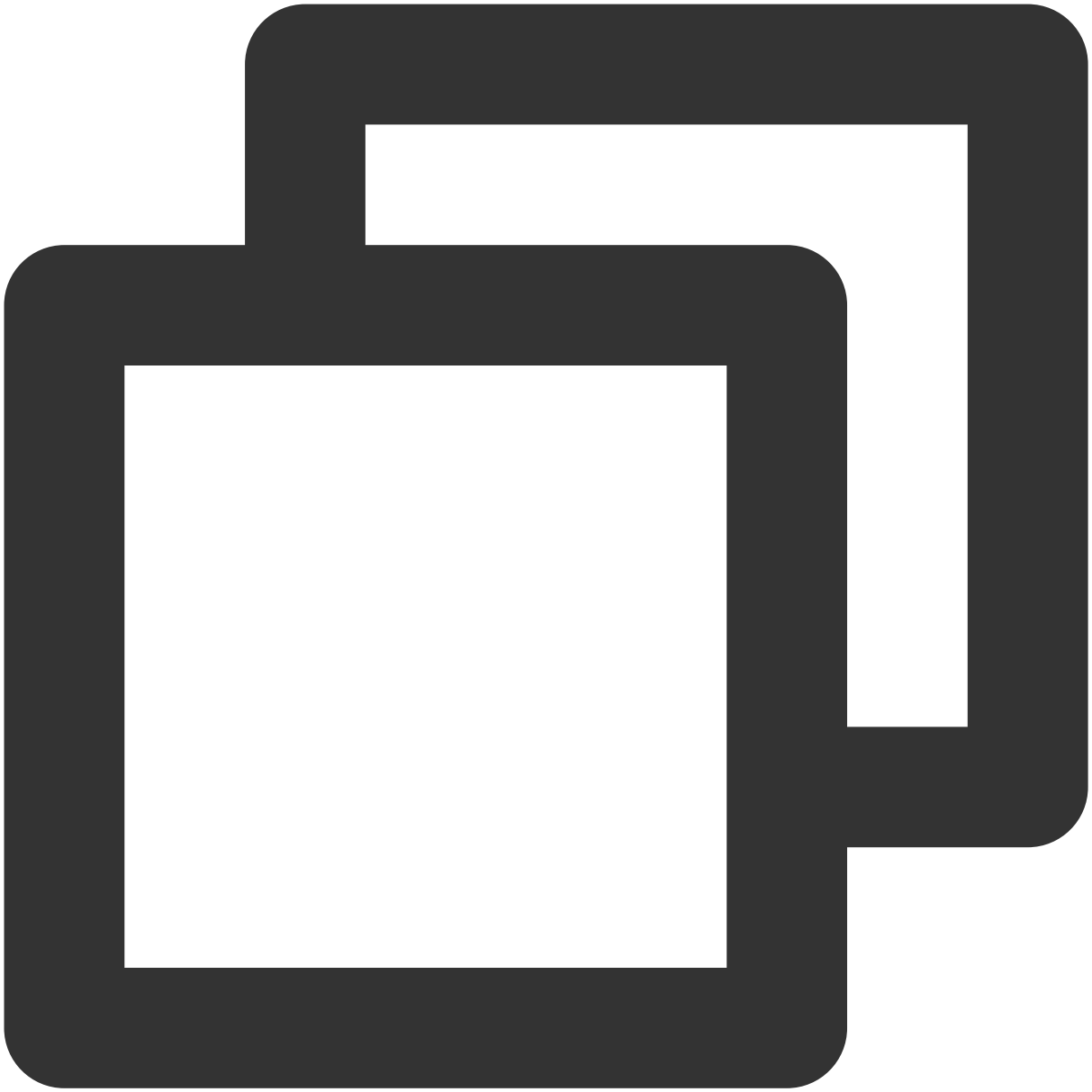
Tencent Cloud's audio and video libraries cannot be integrated at the same time, and there may be symbol conflicts. You can handle it according to the following scenarios.

1. If you are using the `TXLiteAVSDK_TRTC` library, there will be no symbol conflicts. You can directly add dependencies in the Podfile file.



```
pod 'TUILiveKit'
```

2. If you are using the `TXLiteAVSDK_Professional` library, there will be symbol conflicts. You can add dependencies in the `Podfile` file.



```
pod 'TUILiveKit/Professional'
```

3. If you are using the `TXLiteAVSDK_Enterprise` library, there will be symbol conflicts. It is recommended to upgrade to `TXLiteAVSDK_Professional` and then use TUILiveKit/Professional.

How to view TRTC logs?

TRTC logs are compressed and encrypted by default, with the extension .xlog. Whether the log is encrypted can be controlled by setLogCompressEnabled. The file name containing C(compressed) is encrypted and compressed, and the file name containing R(raw) is plaintext.

iOS：Sandbox's `Documents/log`

Note:

To view the .xlog file, you need to download the decryption tool and run it directly in the Python 2.7 environment with the xlog file in the same directory using python decode_mars_log_file.py.

To view the .clog file (new log format after version 9.6), you need to download the decryption tool and run it directly in the Python 2.7 environment with the clog file in the same directory using python decompress_clog.py.

Android

Last updated：2024-05-17 11:48:33

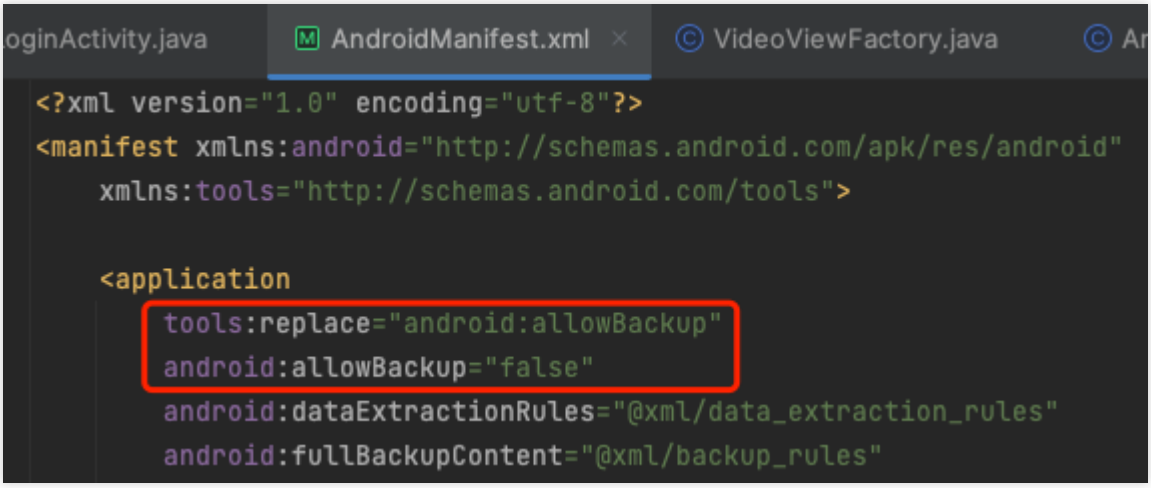
Can TUILiveKit use TRTC without introducing IM SDK?

No, all the components of TUIKit use Tencent Cloud IM SDK as the basic service for communication, such as the core logic of creating room signaling, Lian-mic signaling, etc., all use IM services. If you have purchased other IM products, you can also refer to TUILiveKit logic to adapt.

Exception: allowBackup exception, How to Handle?

```
Manifest merger failed : Attribute application@allowBackup value=(false) from AndroidManifest.xml:7:9-3
    is also present at [com.github.yyued:SVGAPlayer-Android:2.6.1] AndroidManifest.xml:12:9-35 value=(tr
    Suggestion: add 'tools:replace="android:allowBackup"' to <application> element at AndroidManifest.xr
```

Reasons：The `allowBackup` property is configured in the `AndroidManifest.xml` of several modules, causing conflicts.
Solution：You can remove the `allowBackup` attribute from your project's `AndroidManifest.xml` file or change it to false to turn off backup and restore, And add `tools:replace="android:allowBackup"` in the application node of the `AndroidManifest.xml` file; Indicates to override the settings of other modules, using your own Settings.



Activity theme issue: activity need to use a Theme.AppCompat theme, How to Handle?

```
FATAL EXCEPTION: main
Process: com.trtc.uikit.livekit.example, PID: 15190
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.trtc.uikit.livekit.example/com.trtc.u
 .LoginActivity}: java.lang.IllegalStateException: You need to use a Theme.AppCompat theme (or descendant) w
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3730)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3885)
    at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:101)
    at android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135)
    at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2332)
    at android.os.Handler.dispatchMessage(Handler.java:107)
    at android.os.Looper.loop(Looper.java:230)
    at android.app.ActivityThread.main(ActivityThread.java:8115) <1 internal line>
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:526)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1034)
```

Reasons: Since `LoginActivity` inherited from `AppCompatActivity`, a `Theme.AppCompat` was to be given to `LoginActivity`.

Solution：You can add a `Theme.AppCompat` theme to the `LoginActivity` configuration in your project's `AndroidManifest.xml` file. You can also use your own `Theme.AppCompat` theme. An example of a fix is shown in the image:

```
<activity
    android:name=".login.LoginActivity"
    android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
</activity>
```