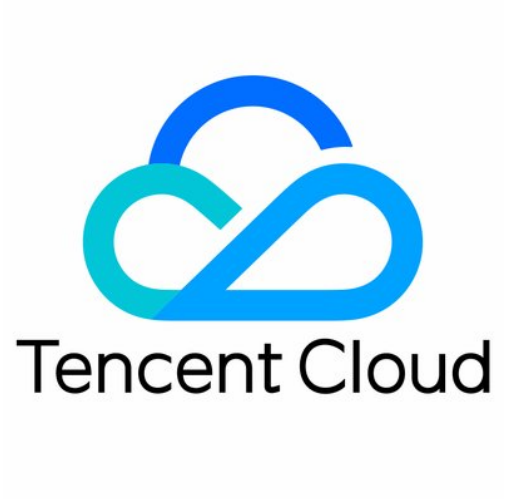
Tencent Real-Time Communication

Live Streaming (Including UI)

Product Documentation



Contents

Live Streaming (Including UI) Overview (TUILiveKit)

Last updated：2024-08-15 14:42:54

Overview

TUILiveKit enables interactive live streaming for scenarios such as social entertainment, shopping, and fitness classes. You can quickly add in-room communication, gift sending, room management, and other features to your app with just three steps in as little as 30 minutes. The diagram below shows the basic features of the component.

Supported Platforms

Platform	Android	iOS	Desktop	Flutter
Supported				
Supported Languages/Frameworks	Java Kotlin	Swift Objective-C	Electron(Only **Windows** currently)	Dart

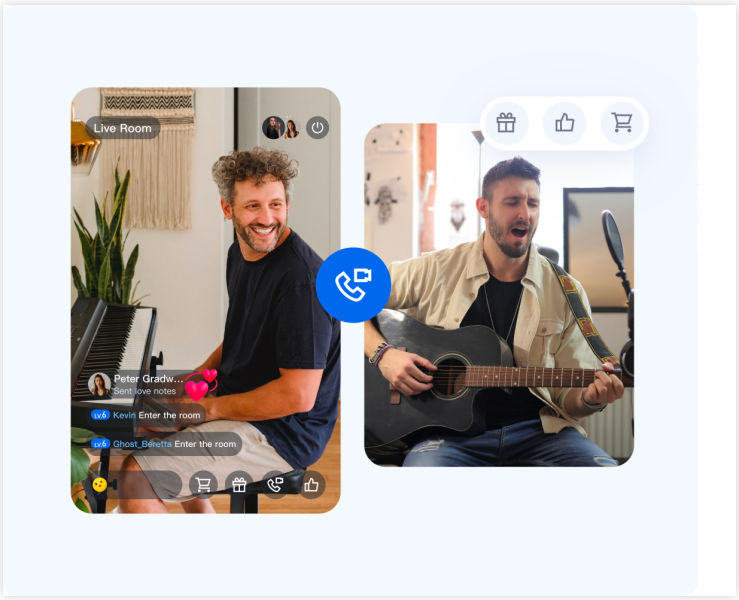
Features

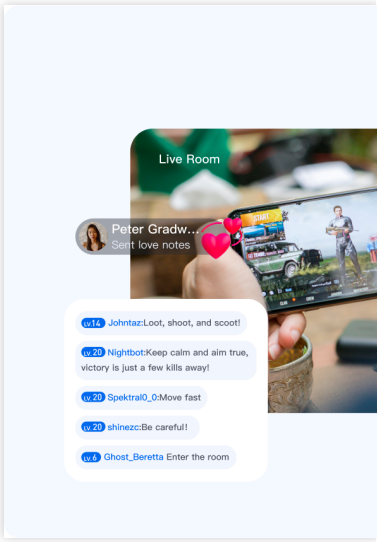
Basic Feature	Advanced Feature	Strengths
HD Live Streaming Voice Chat Room Live Viewing Viewer Mic Connection Viewer List Member Management	On-Screen Commenting Liking Interactive Gifts (Fullscreen Gifts) Sound Effects & Voice Changer	**Comprehensive UI Interaction** **AI Super Resolution** **Professional Live Streaming 3A Algorithm, Better Audio Quality** **Rich REST APIs**

Use Cases

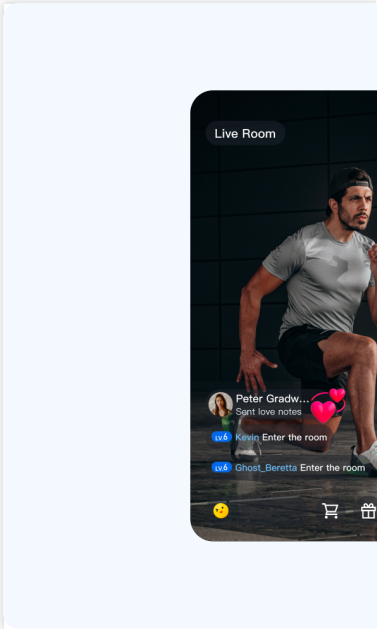
TUILiveKit is suitable for all kinds of high-concurrency and large-scale live streaming scenarios such as live show, live shopping, live sports streaming, live product launch, live roadshow, and online auction.

Social entertainment	Game Interaction





Live shopping	Fitness communication



Trying It Online

Platform	Android	iOS	Desktop	Flutter

Demo Integration	Github: Andorid	Github: iOS	GitHub: Electron	Github: Flutter

Suggestions and Feedback

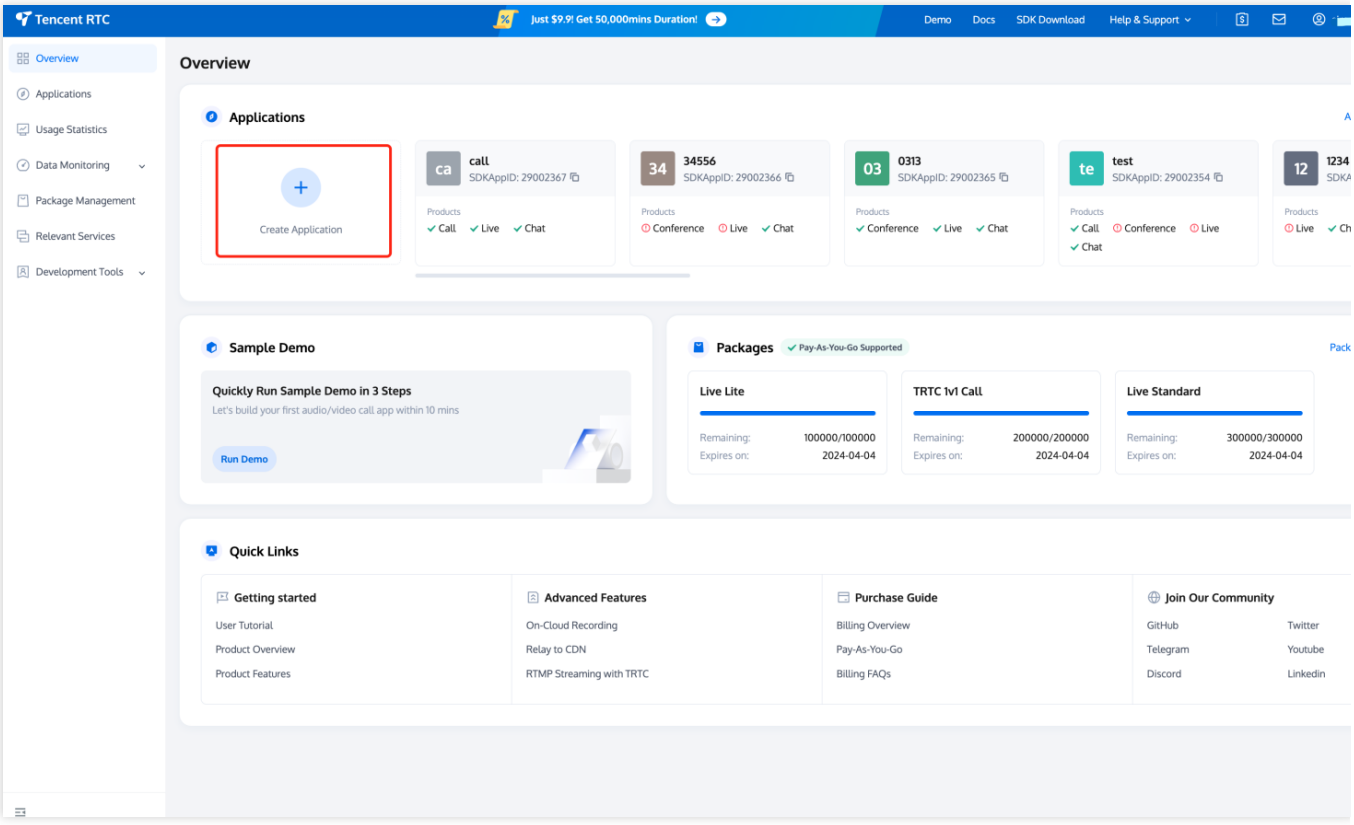
If you have any requirements or feedback, you can contact: info_rtc@tencent.com.

Activating the Service (TUILiveKit)

Last updated：2024-07-03 17:01:14

Free trial

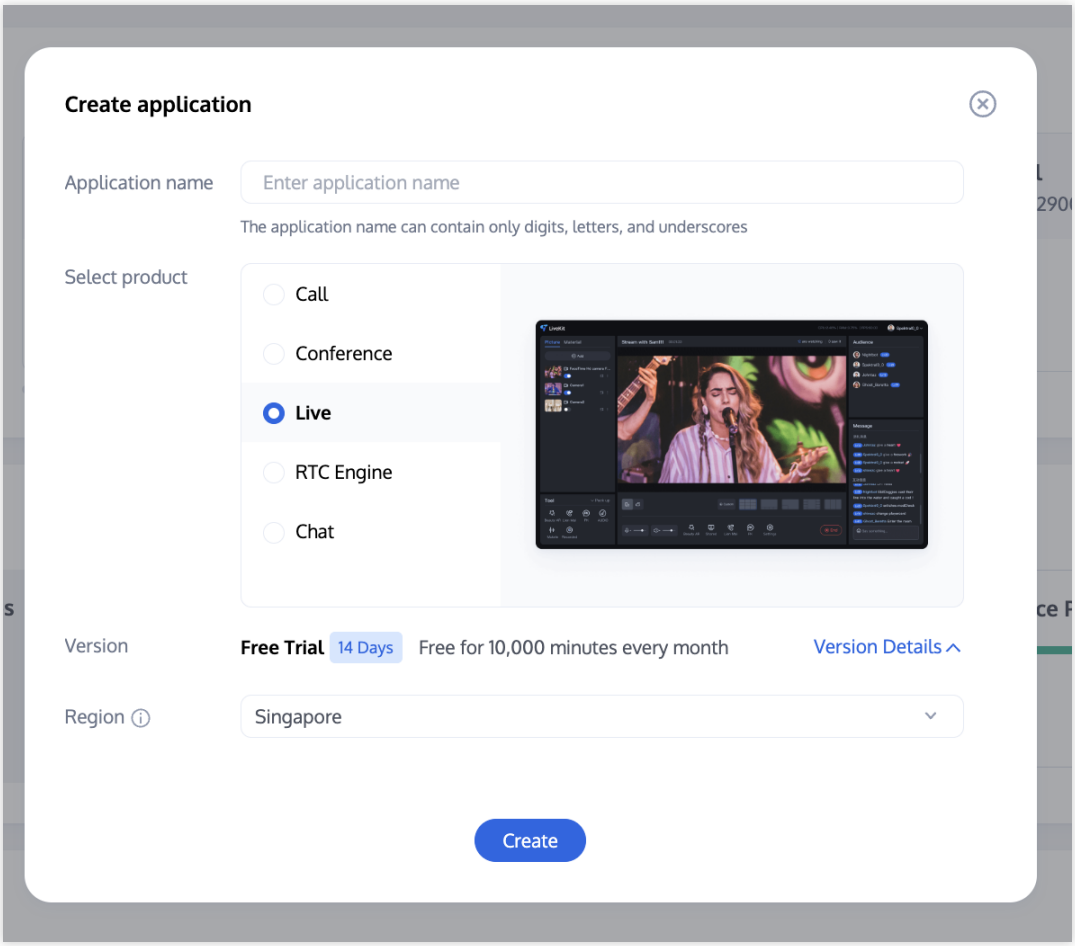
In order for you to better experience the features of TRTC Live, we provide a 14-day free trial. Each SDKAppID can try TRTC Live twice for free, each time for 14 days. Each account can try out TRTC Live 10 times in total.
You can refer to the following guidelines to activate the trial edition of Live.
1. Log in to the TRTC Console and click **Create Application**.



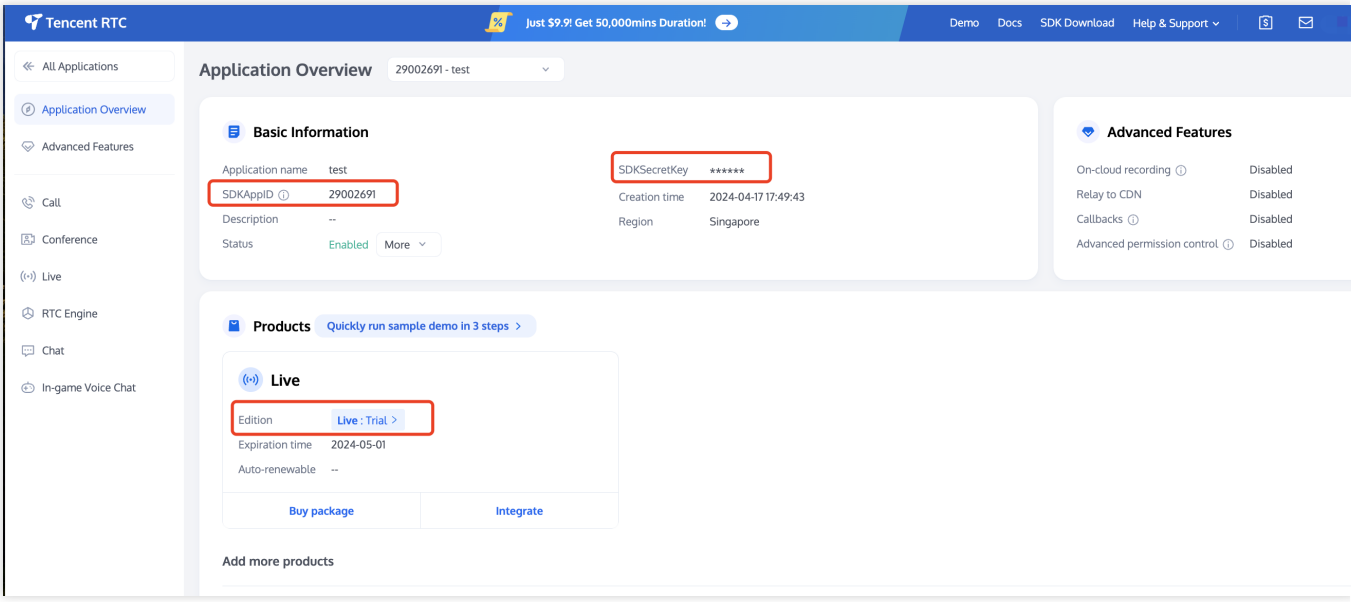
2. In the popup, enter an **application name**, select **Live**, and choose the appropriate data storage **region**. Then click **Create**. This will create a TRTC application bound to the trial edition of TRTC Live.

Note：

By default, all TRTC data is stored in Singapore, while Chat data is stored in the data center you select.



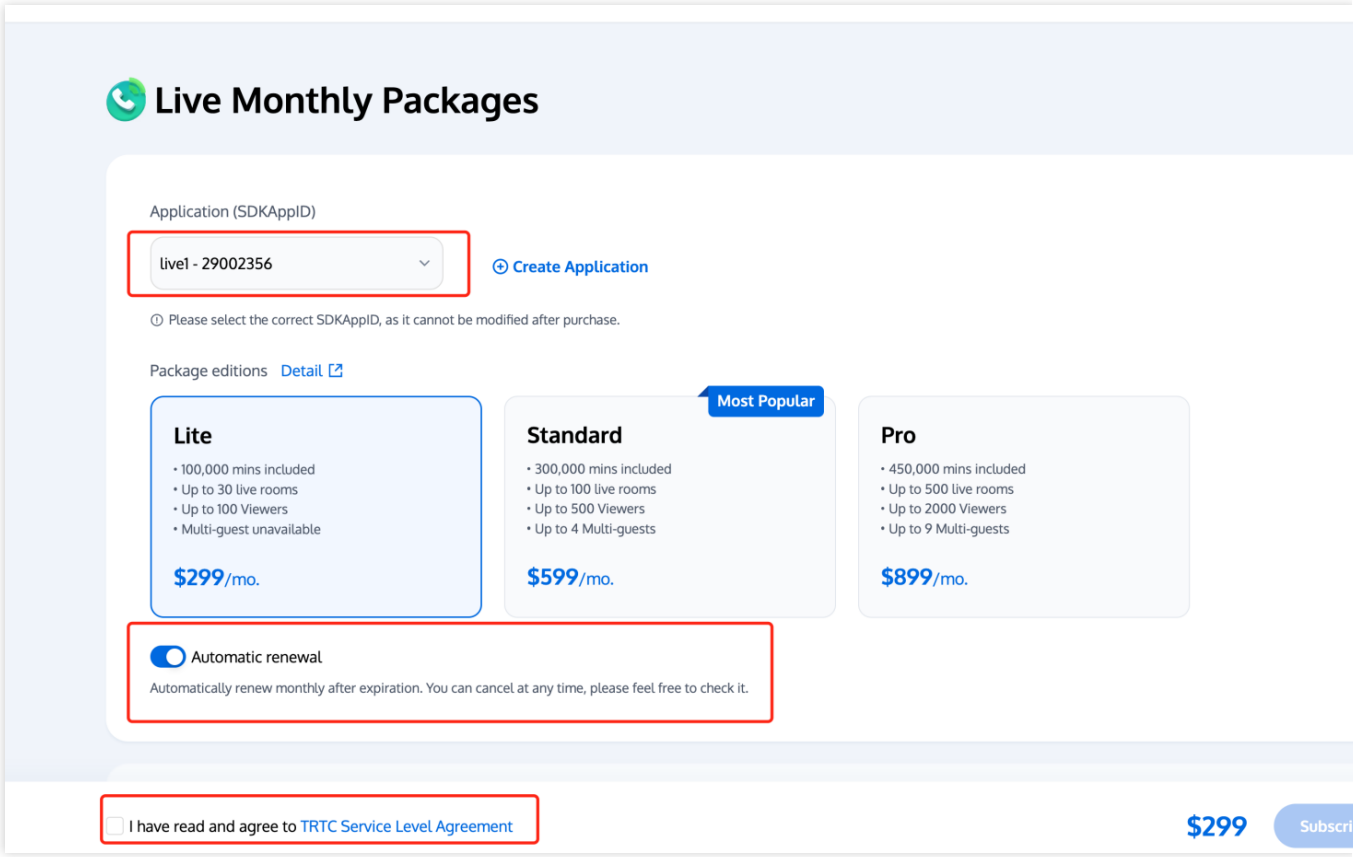
3. On the Application Overview page, you can view the application's `SDKAppID` and `SDKSecretKey`, which will be used in the following steps.



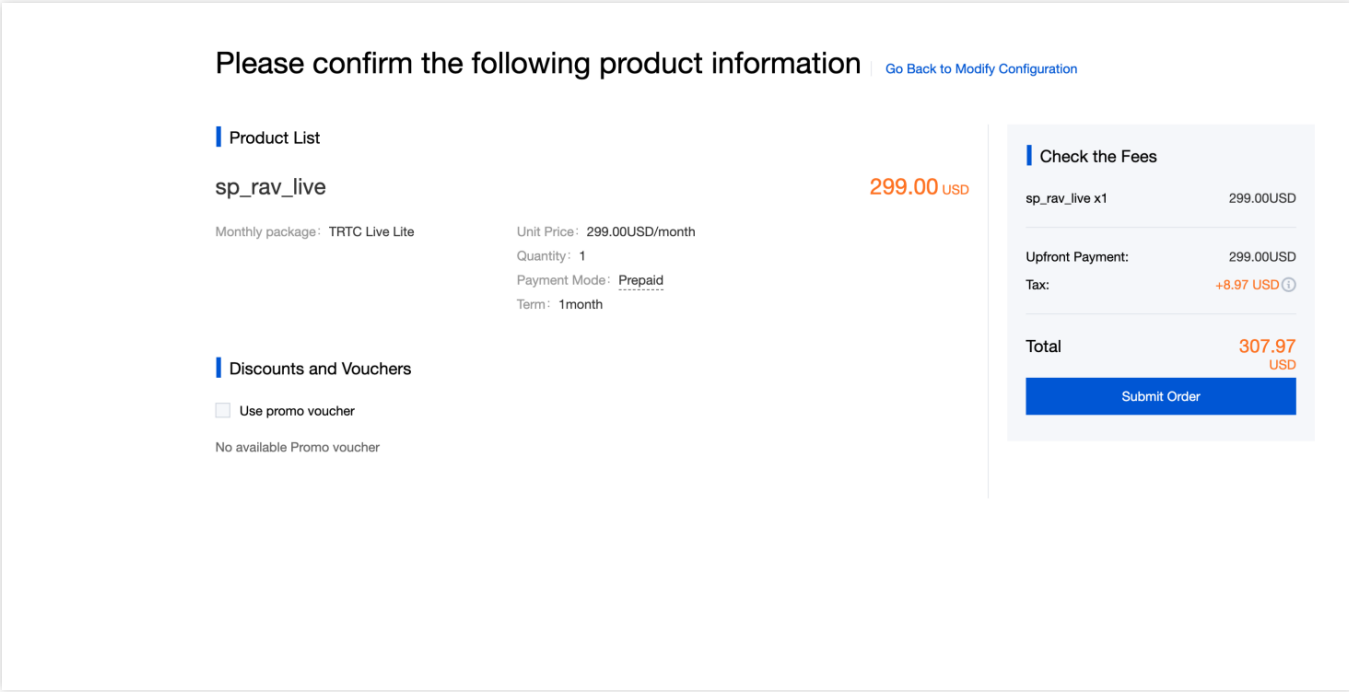
Purchase an official package

You need to buy a TRTC Live monthly package in order to use TRTC Live features. For more information about the pricing and features of different editions, see here.

1. Visit the purchase page, select an application (SDKAppID), and choose the package you want to purchase. **We recommend you enable auto-renewal to avoid business interruptions**. After confirming the purchase information, check the agreement and click **Subscribe now**.



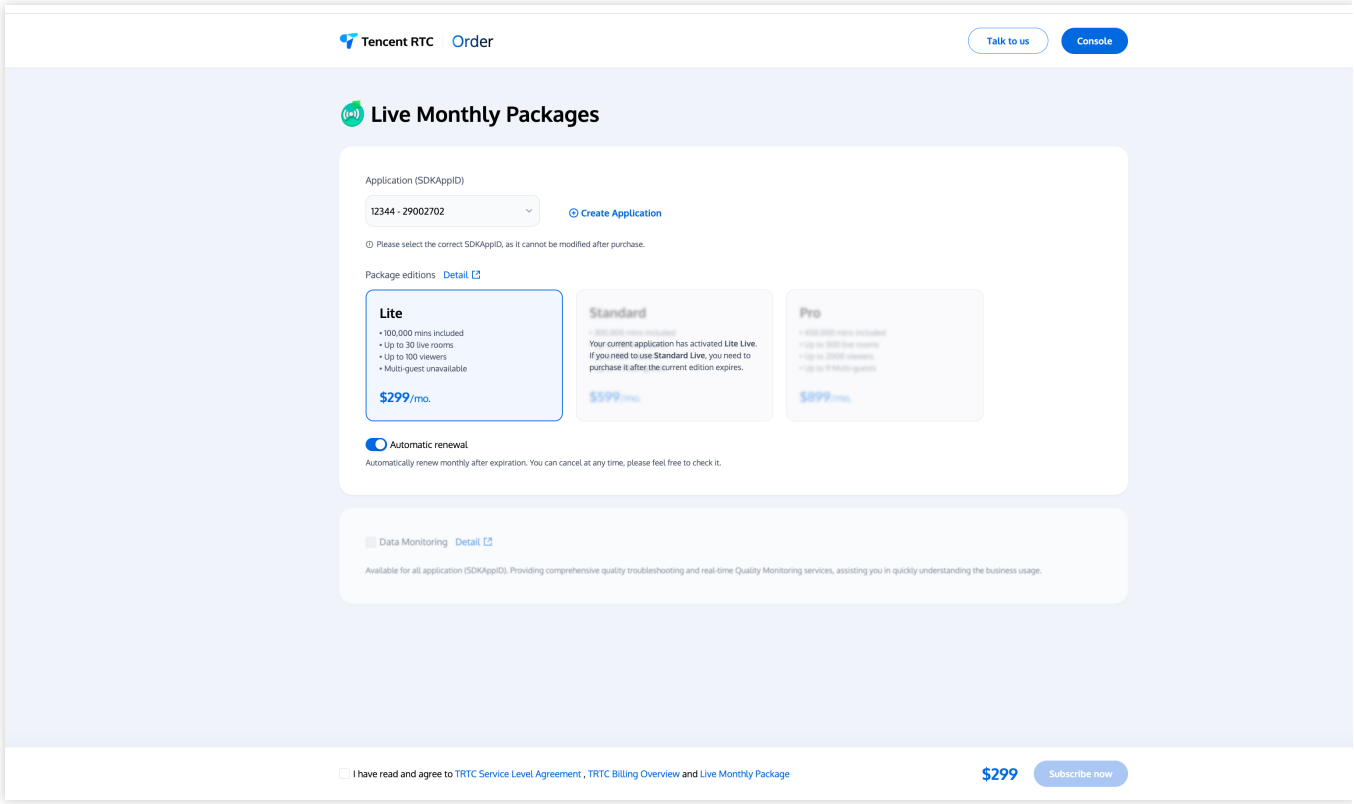
2. Go to the order confirmation page to confirm the product information.



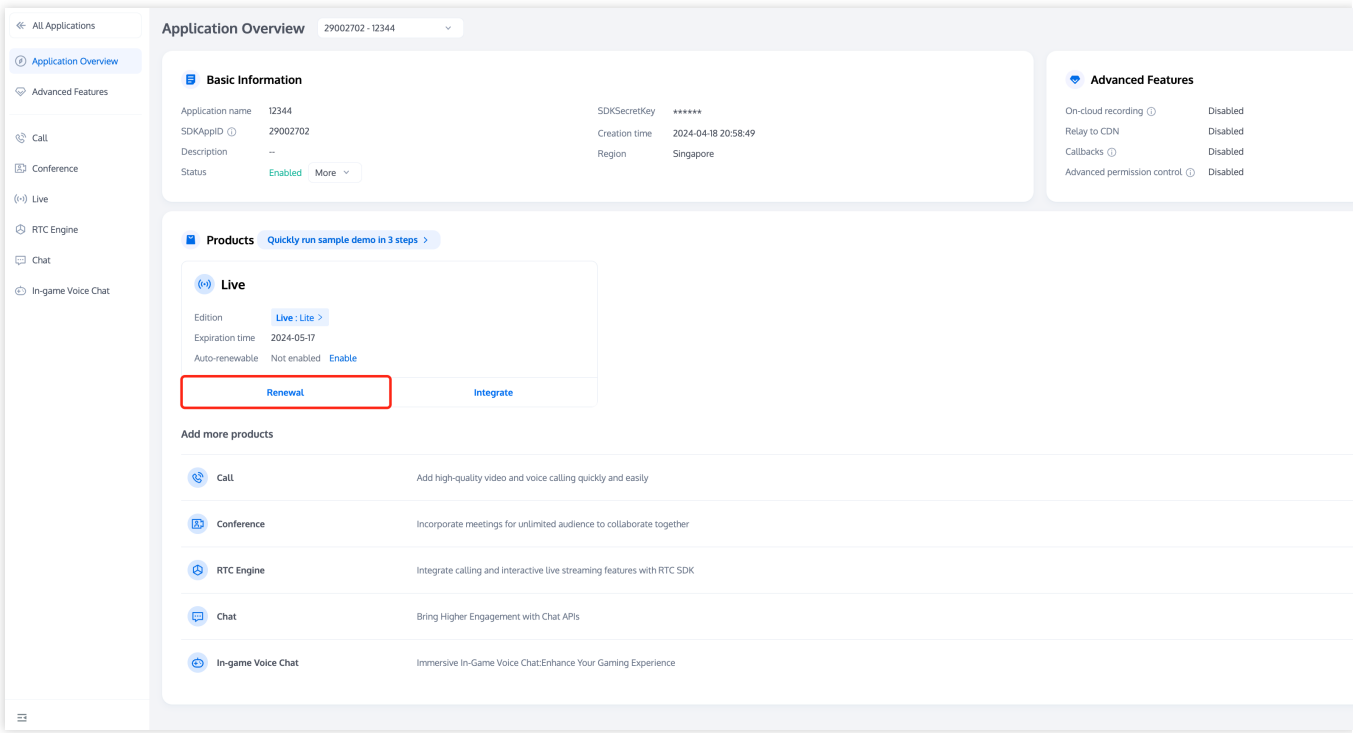
3. Go to the payment page to complete the payment. Once your purchase is complete, you can go to the TRTC console to view the package edition and refer to Integration Guide to integrate the component.

Renew an official package

To renew an official package, simply repeat the steps in Purchase an official package to buy a package of the same edition. **Make sure you select the correct SDKAppID**. We recommend you enable auto-renewal so that your package automatically renews monthly (as long as your account has sufficient balance).

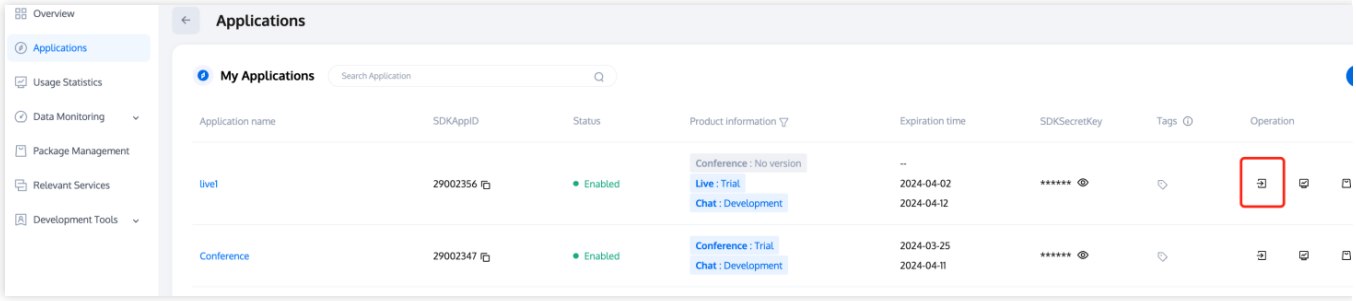


You can also click **Renewal** in the console to renew your package.

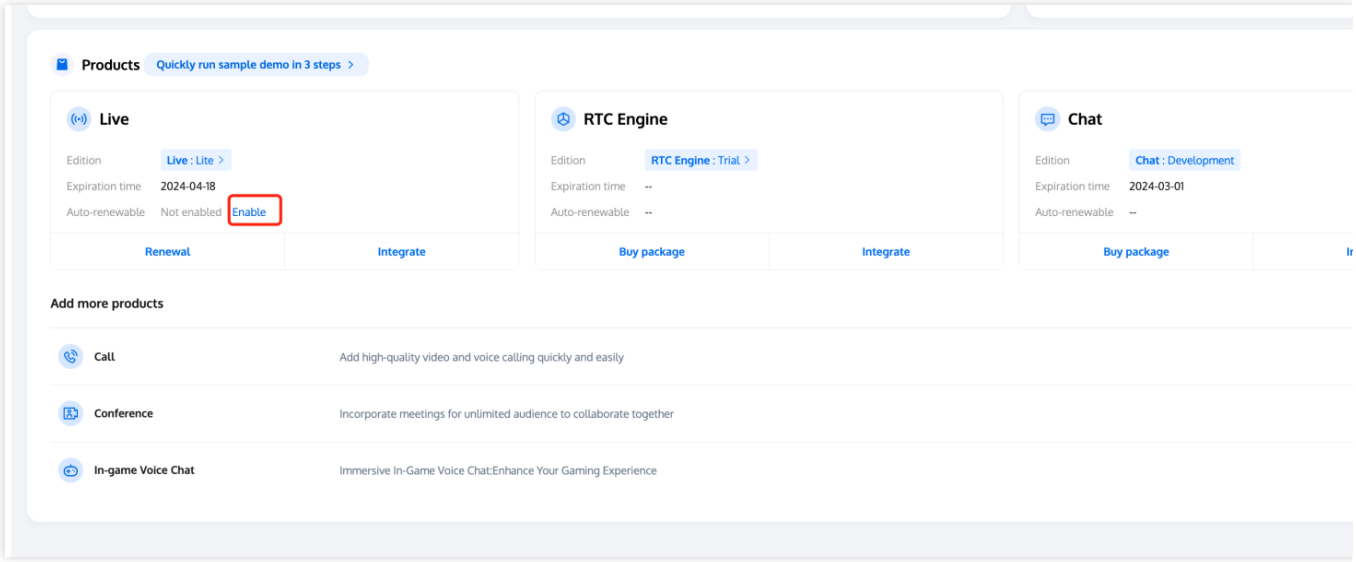


The steps to enable auto-renewal in the console are as follows:

1. Access TRTC Console > Applications, select the application you want to enable auto-renewal for and click the **Manage** button to enter the application details page.



2. Find the information card for Live and click **Enable for Auto-renewable**. In the pop-up window, click **Enable**.

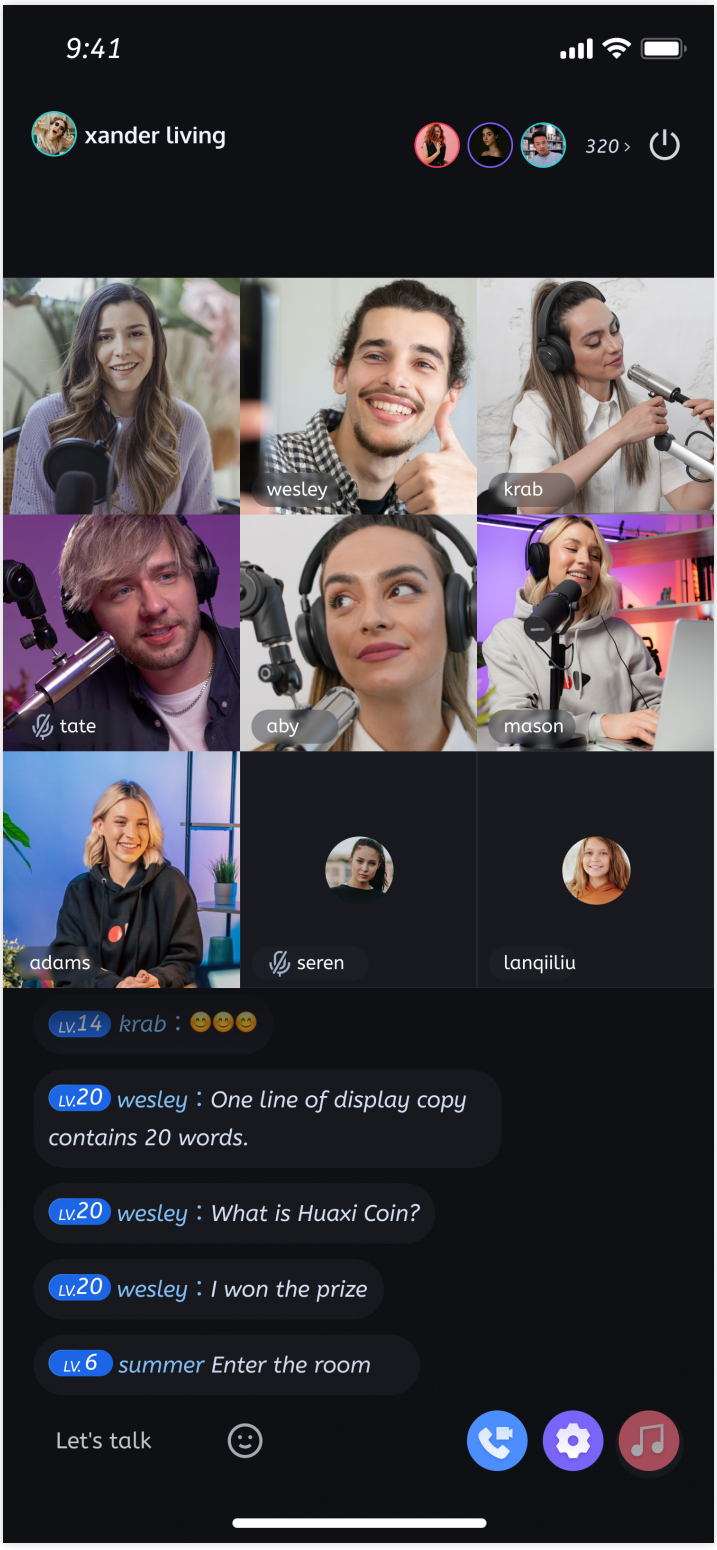


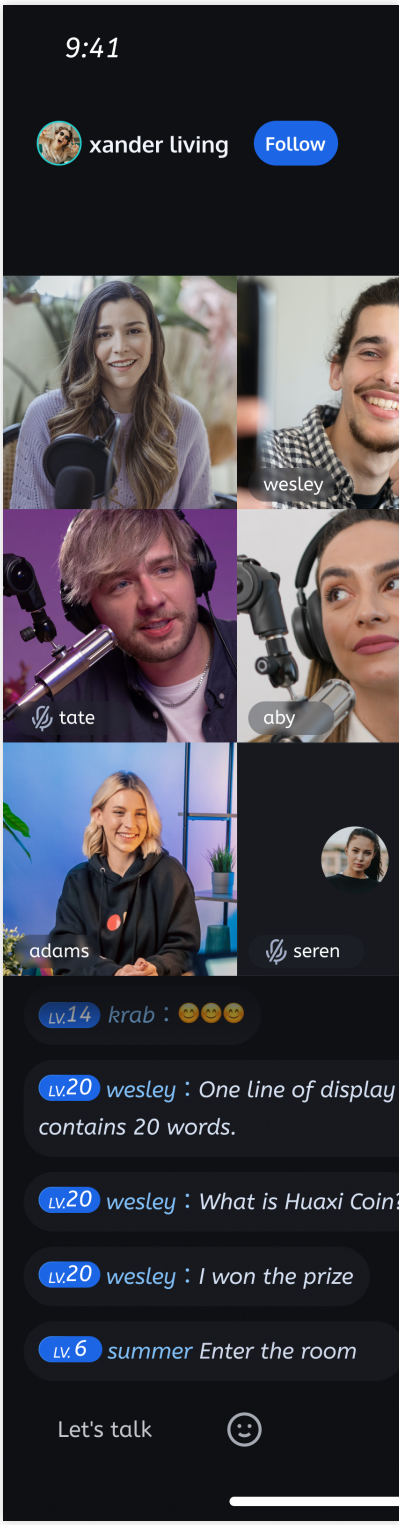
Run Demo（TUILiveKit）
iOS

Last updated：2024-06-24 15:50:37

This article will show you how to quickly run the video live streaming demo. Following this document, you can run the demo in 10 minutes and finally experience a video live streaming function with a complete UI interface.

Anchor	Audience





Environment Preparations

Xcode 15 or later.

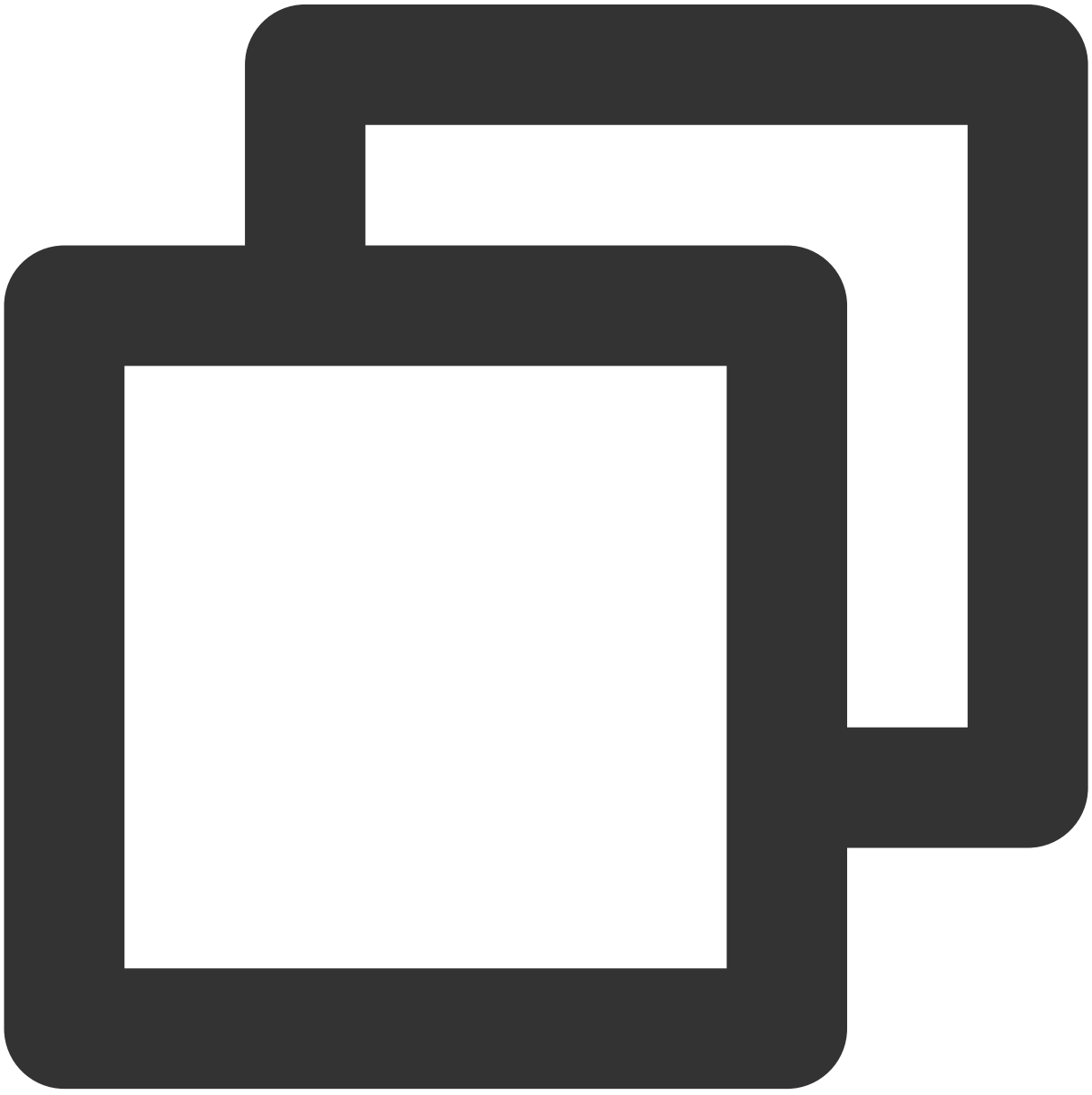
iOS 13.0 or later.

CocoaPods environment installation, click to view.

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

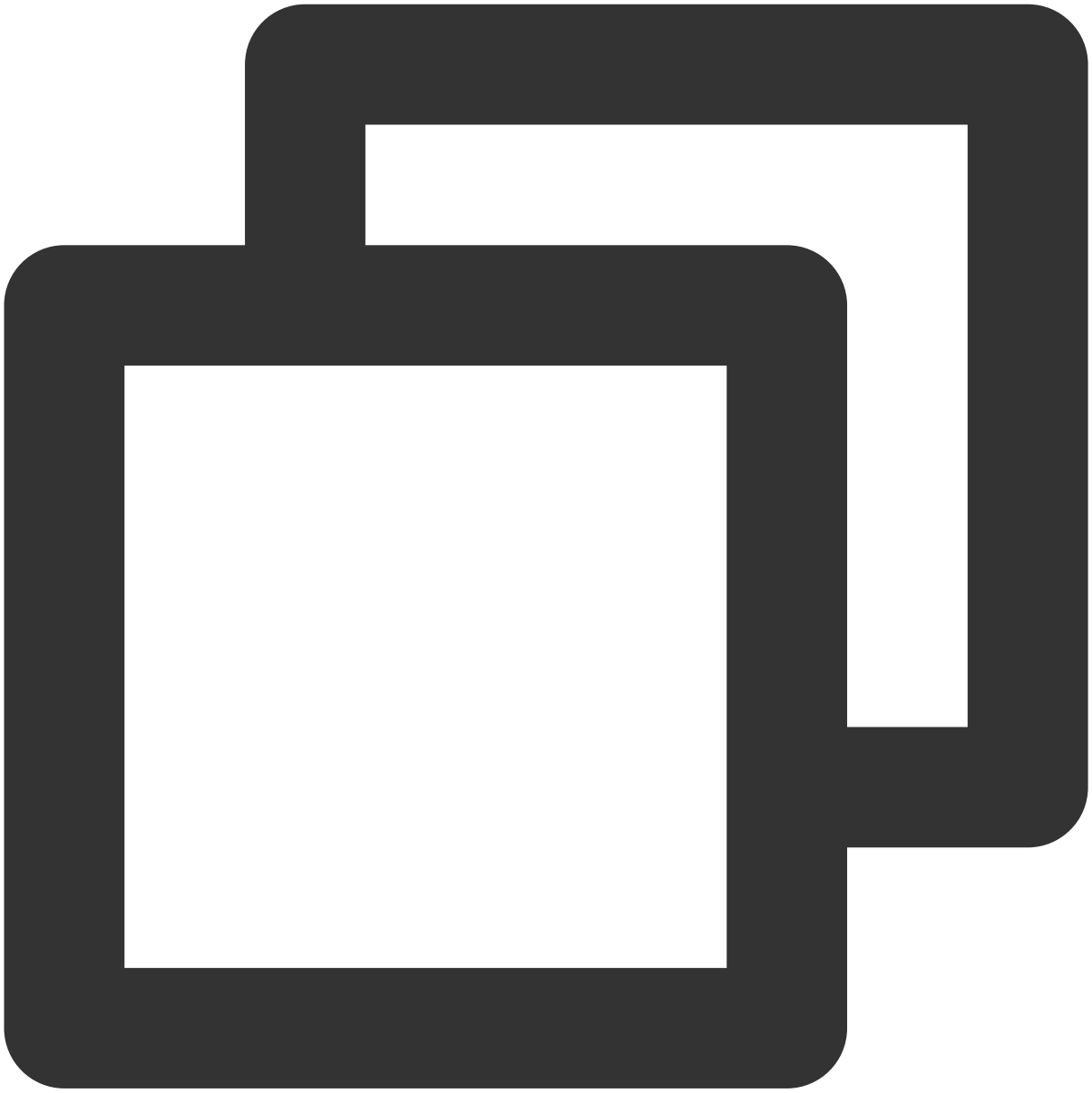
Step 1: Download the Demo

1. Download the TUILiveKit Demo source code from GitHub, or directly run the following command in the command line:



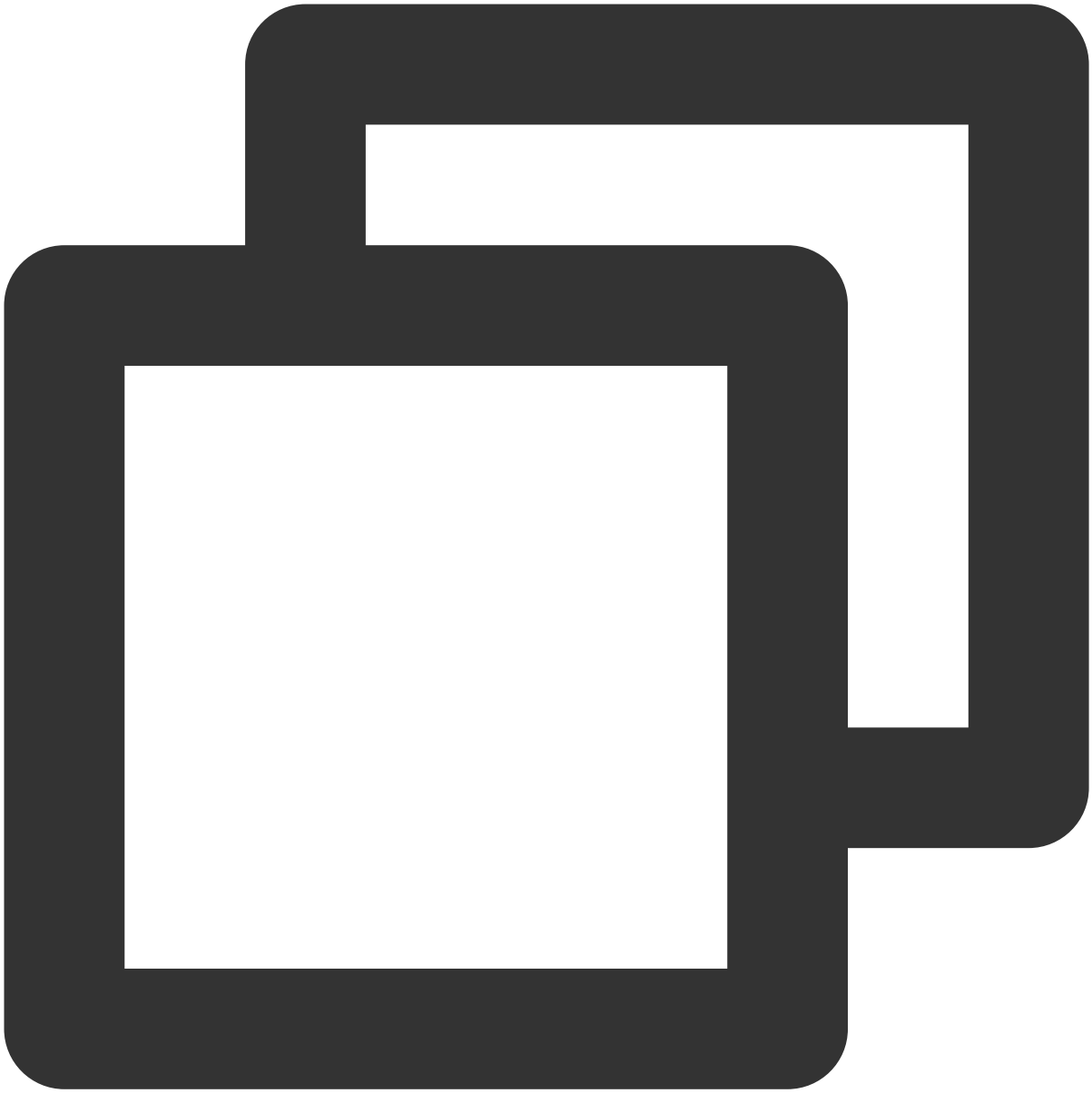
```
git clone https://github.com/Tencent-RTC/TUILiveKit.git
```

2. Go to the iOS project directory on the command line：



```
cd TUILiveKit/iOS/Example
```

3. Load dependent library：



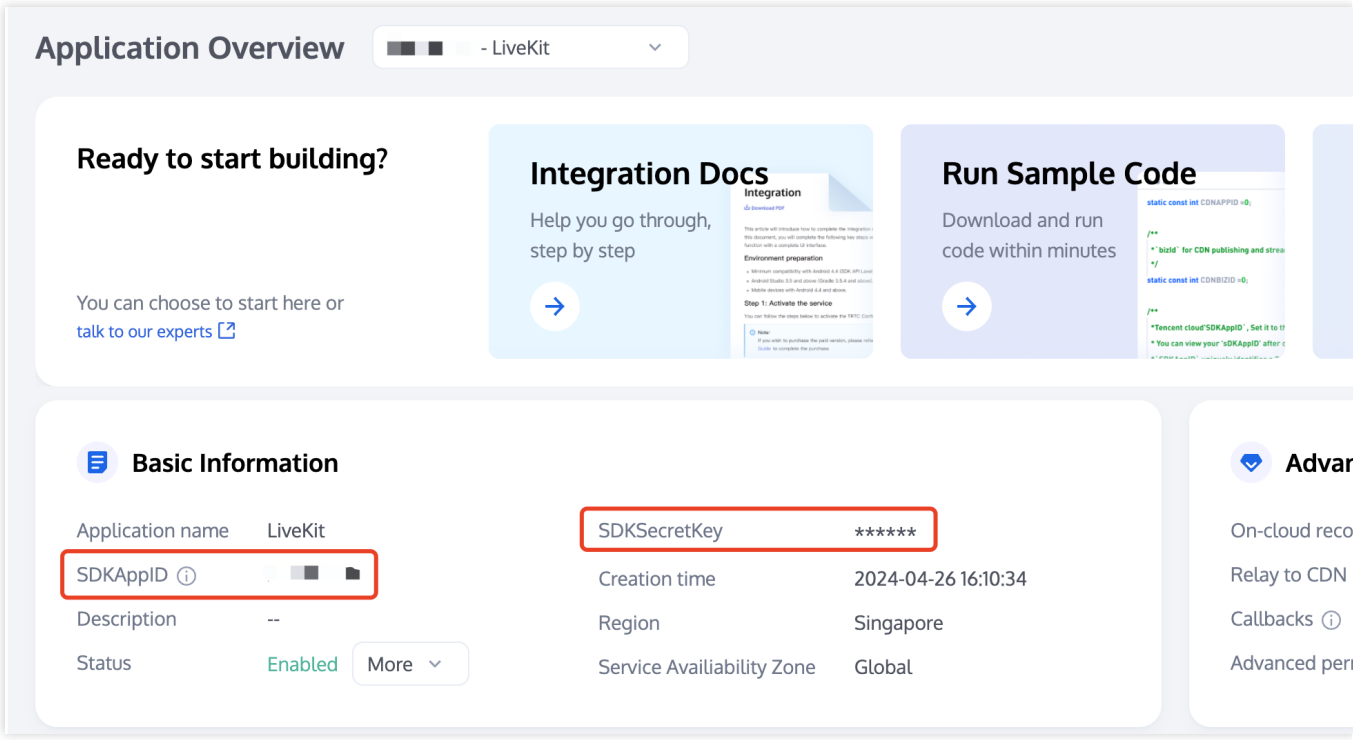
```
pod install
```

Note:

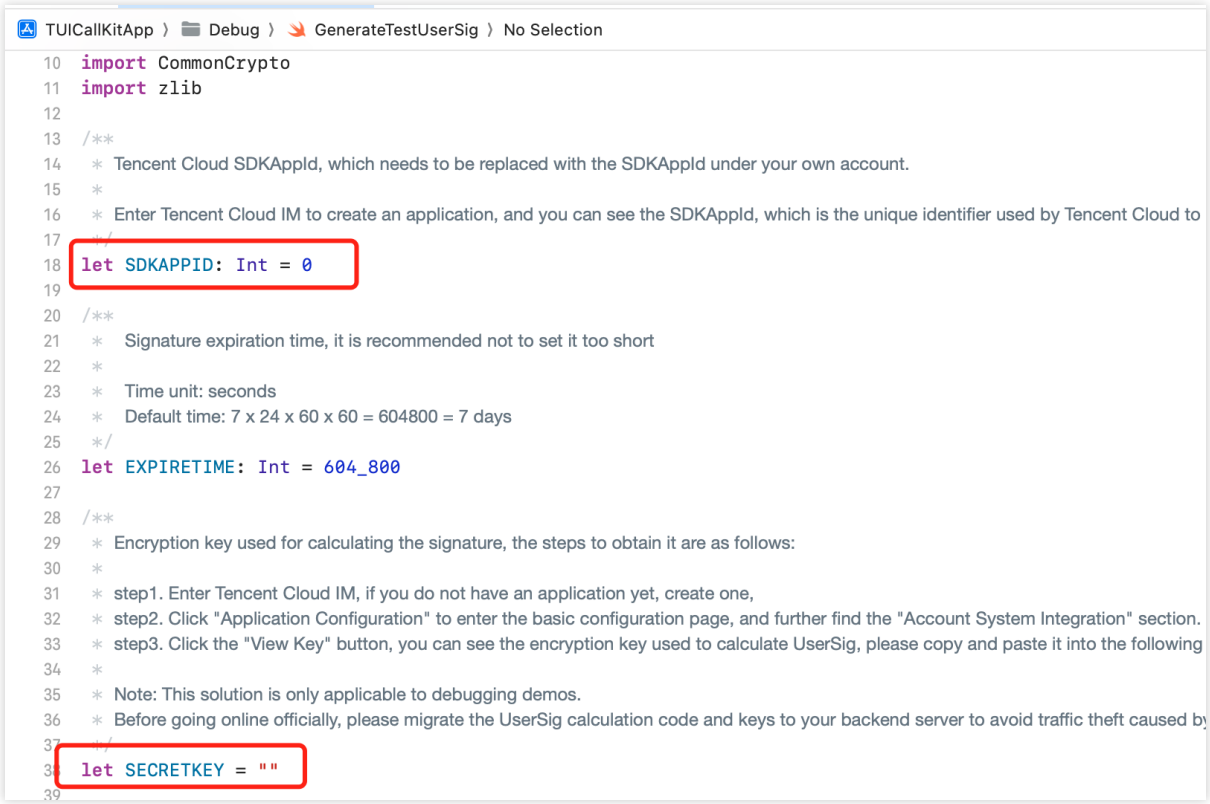
If you do not already have CocoaPods installed, you can learn how to install it here.

Step 2: Configure Demo

1. Activate the TRTC service, obtain the SDKAppID and SDKSecretKey.

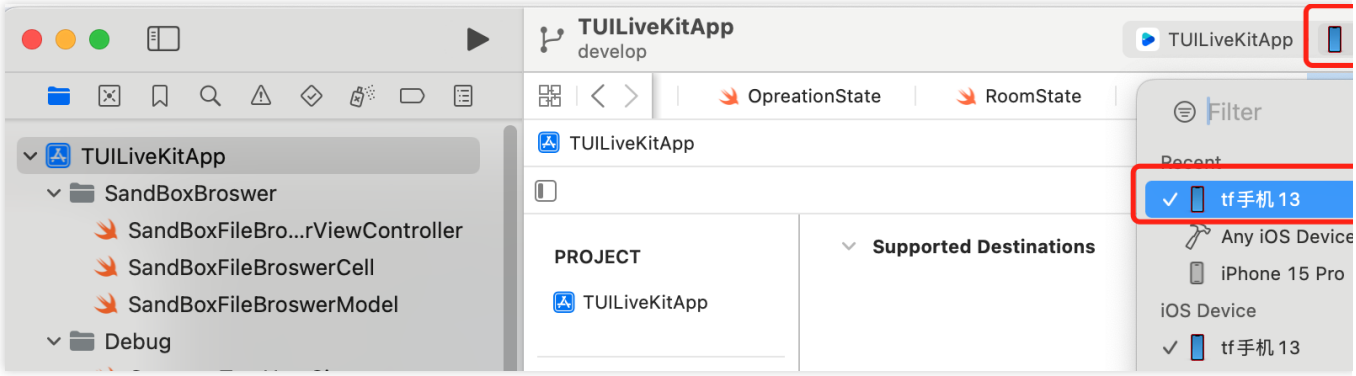


2. Open the `/iOS/Example/Debug/GenerateTestUserSig.swift` Document, file, and enter the SDKAppID and SDKSecretKey obtained when Activate the service:



Step 3: Run the Demo

1. Select the device on which you want to run the Demo in XCode as shown below:



2. Once the selection is complete, click Run to run our Demo on the target device.

3. After Demo runs successfully on the device, you can perform the following steps to initiate and watch live broadcasts.

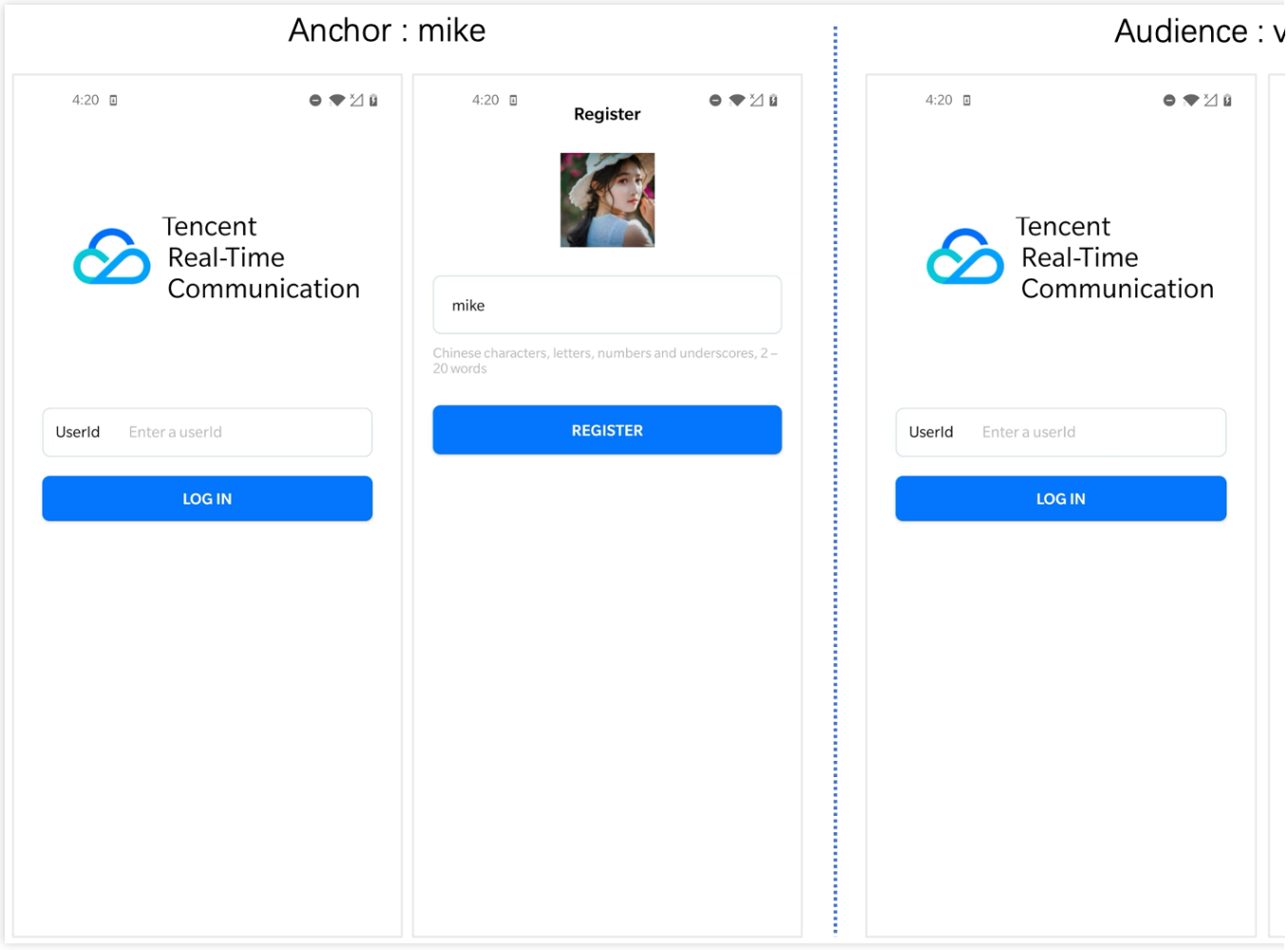
Start Live Broadcast

Note：

In order to allow you to experience the complete video live broadcast process, please log in two users on two devices to use the Demo, one as the host and the other as the audience.

1. Log in & Signup

Please enter your UserId in the User ID field. If your current User ID has not been used before, you will be taken to the Registration page where you can set an avatar and nickname for yourself.

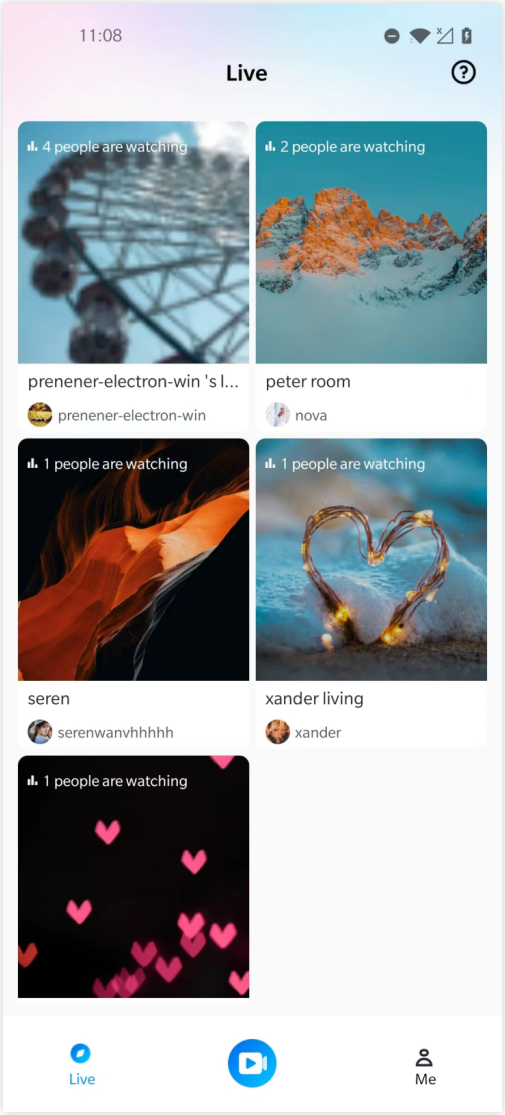


Note：

Try to avoid setting your User ID to simple strings like "1", "123", "111", as TRTC does not support the same User ID being logged into from multiple devices. Such User IDs like "1", "123", "111" are easily occupied by your colleagues during collaborative development, leading to login failures. Therefore, we recommend setting highly recognizable User IDs while debugging.

2. The anchor starts the live broadcast.

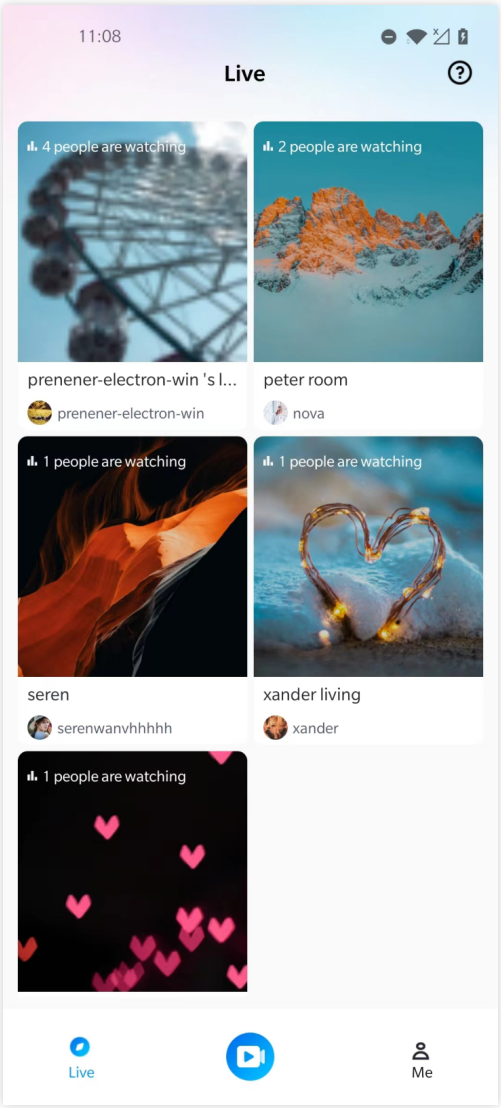
Click the **Start Broadcast** button in the middle of the bottom of the homepage to enter the broadcast preview page, and then click "Start Live" to start the live broadcast.

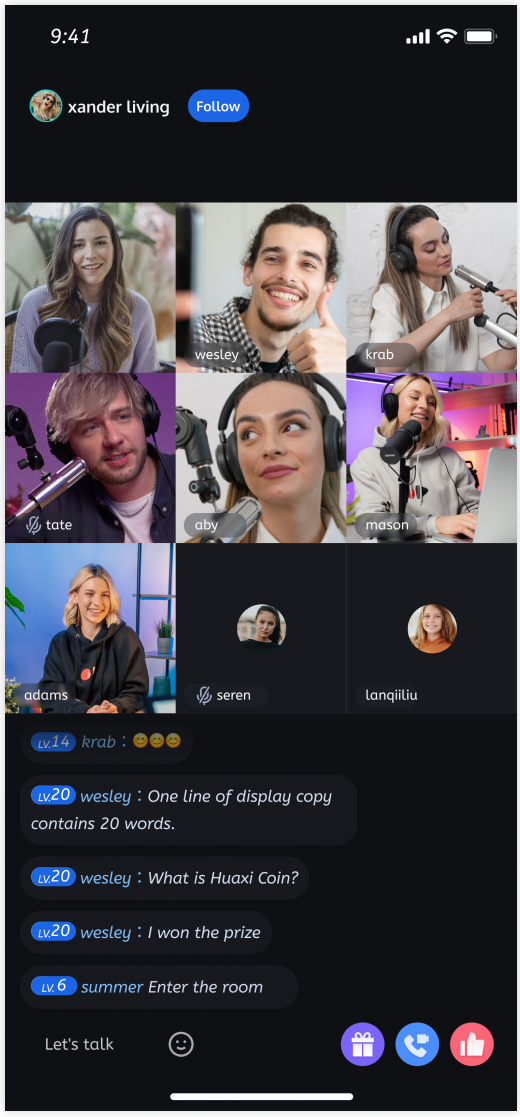
Anchor: Before entering	Anchor: Preview
	

3. Viewers join the live broadcast room.

Click on any room in the live broadcast list to enter the live broadcast room.

Audience: Before entering	Audience: After entering


11:08
Live
4 people are watching
2 people are watching
prenener-electron-win 's l...
prenener-electron-win
peter room
nova
1 people are watching
1 people are watching
seren
serenwanvhhhhh
xander living
xander
1 people are watching
Live
Me

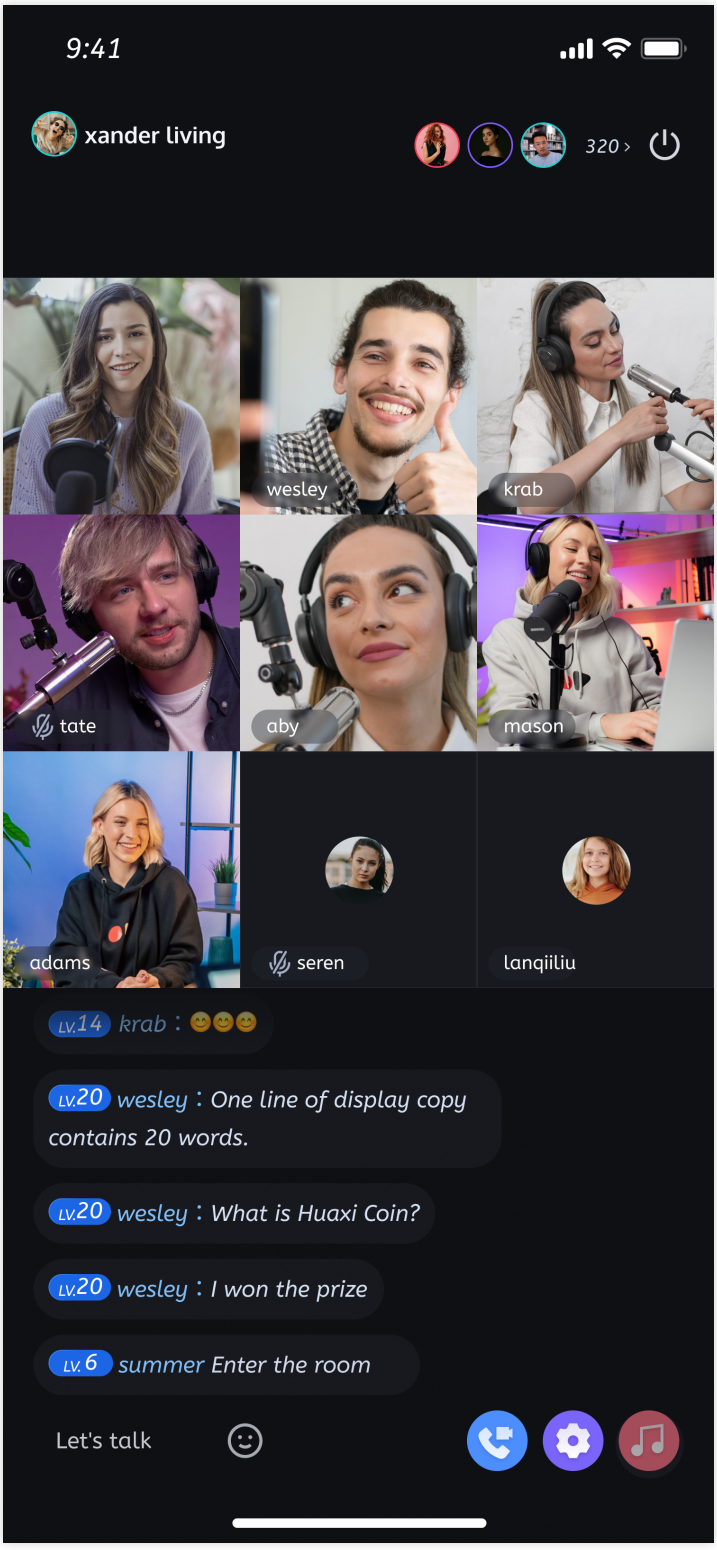

9:41
xander living
Follow
wesley
krab
tate
aby
mason
adams
seren
lanqiiliu
LV14 krab : 😊😊😊
LV20 wesley : One line of display copy contains 20 words.
LV20 wesley : What is Huaxi Coin?
LV20 wesley : I won the prize
LV 6 summer Enter the room
Let's talk

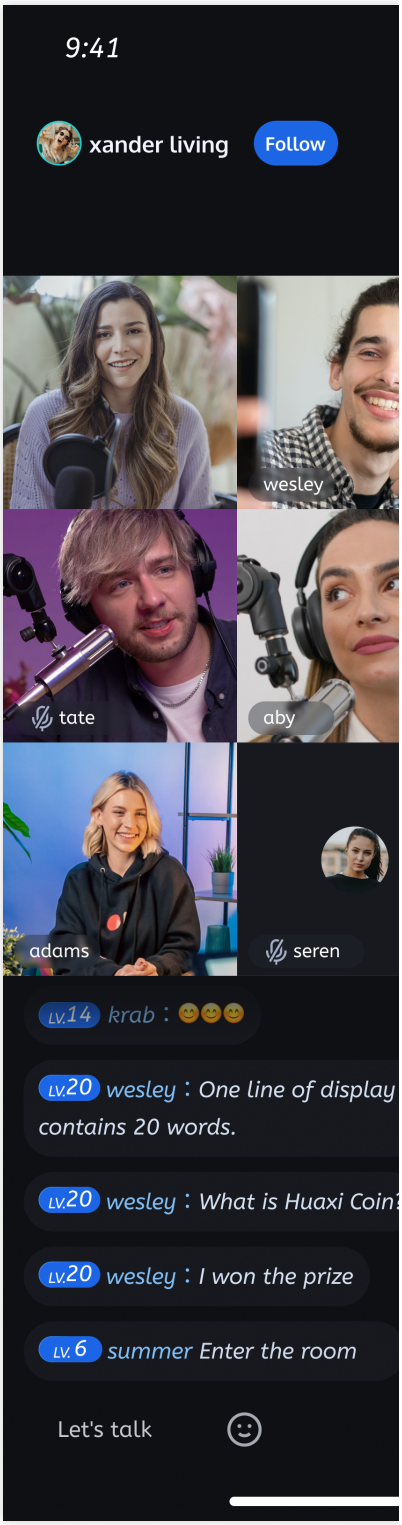
Android

Last updated：2024-08-09 22:25:01

This article will show you how to quickly run the video live streaming demo. Following this document, you can run the demo in 10 minutes and finally experience a video live streaming function with a complete UI interface.

Anchor	Audience





Environment preparations

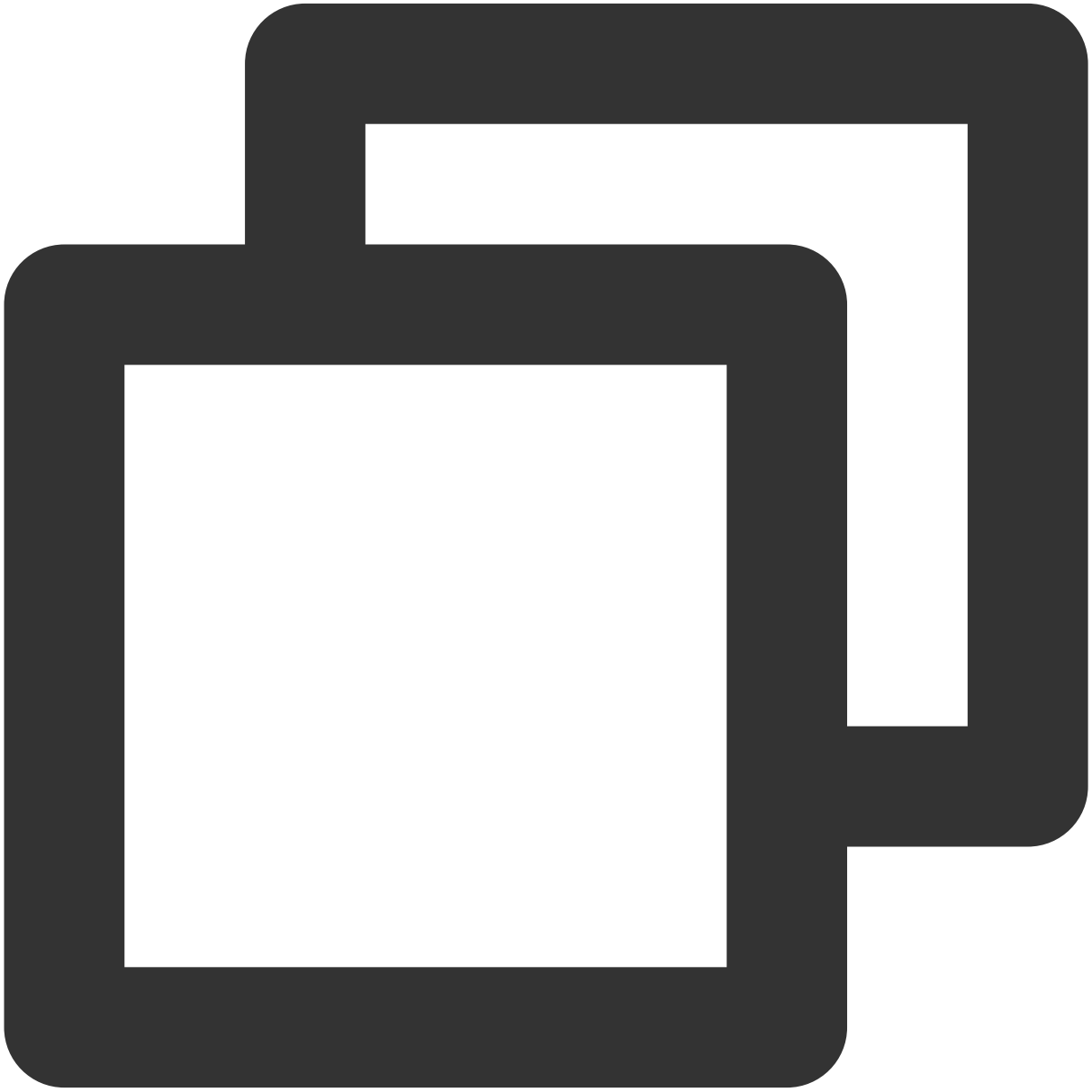
Android 5.0 (SDK API level 21) or above.

Gradle 8.0 or later.

Two Android 5.0 or newer devices.

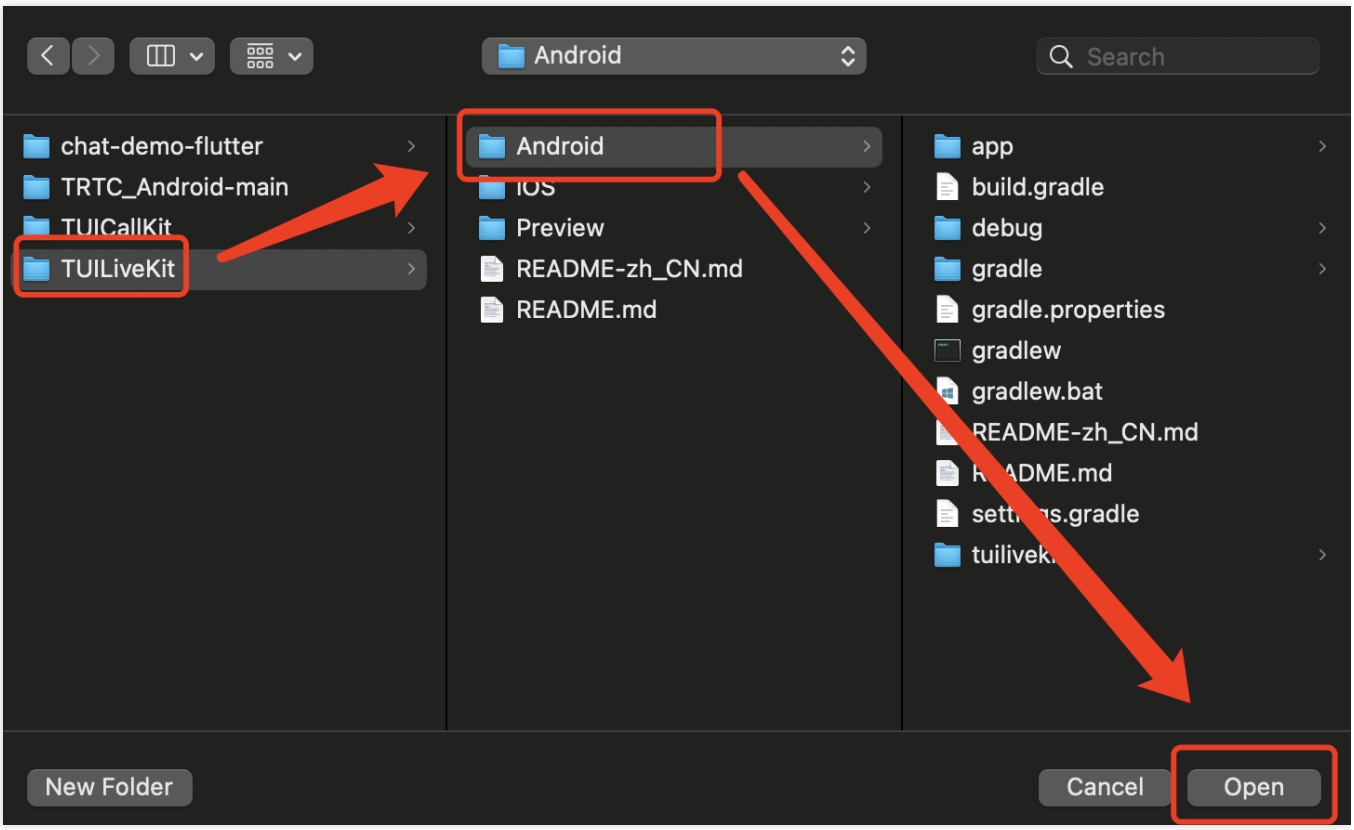
Step 1: Download the Demo

1. Download the TUILiveKit Demo source code from GitHub, or directly run the following command in the command line:



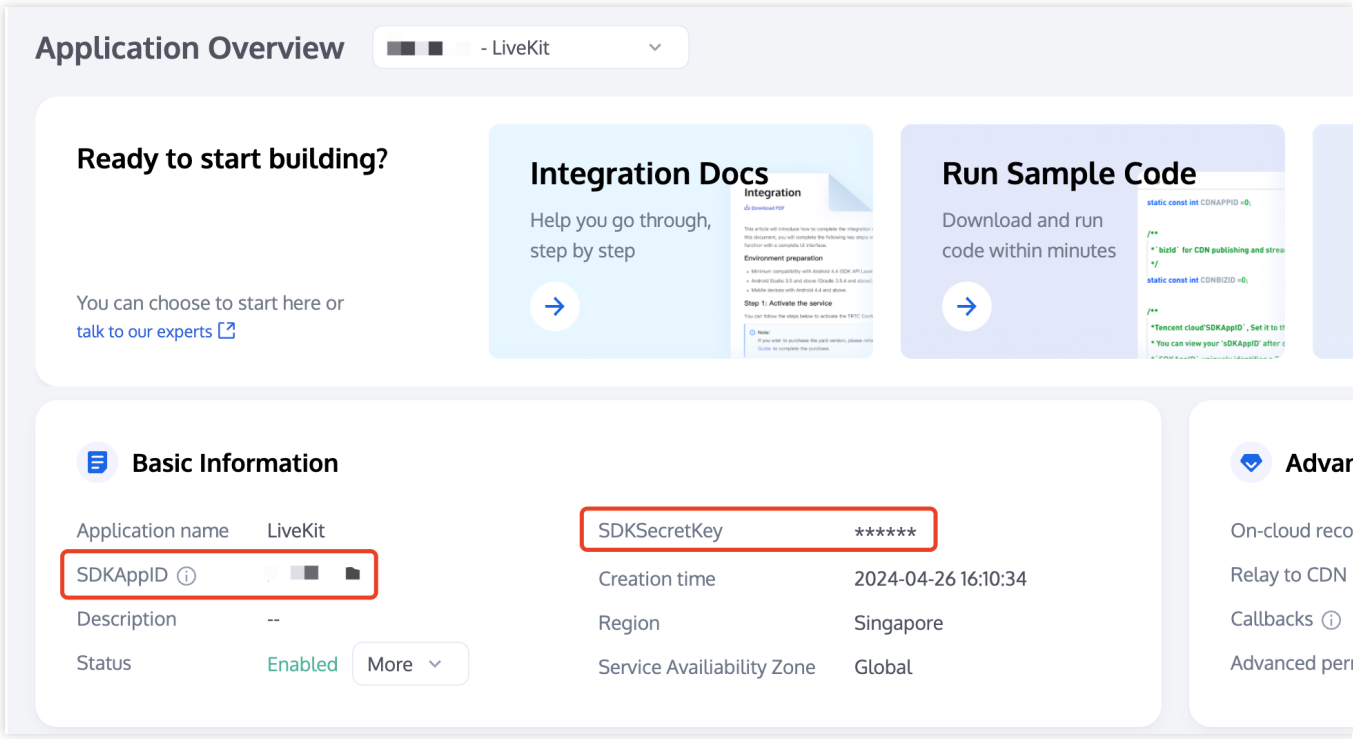
```
git clone https://github.com/Tencent-RTC/TUILiveKit.git
```

2. Open the TUICallKit Android project through Android Studio:

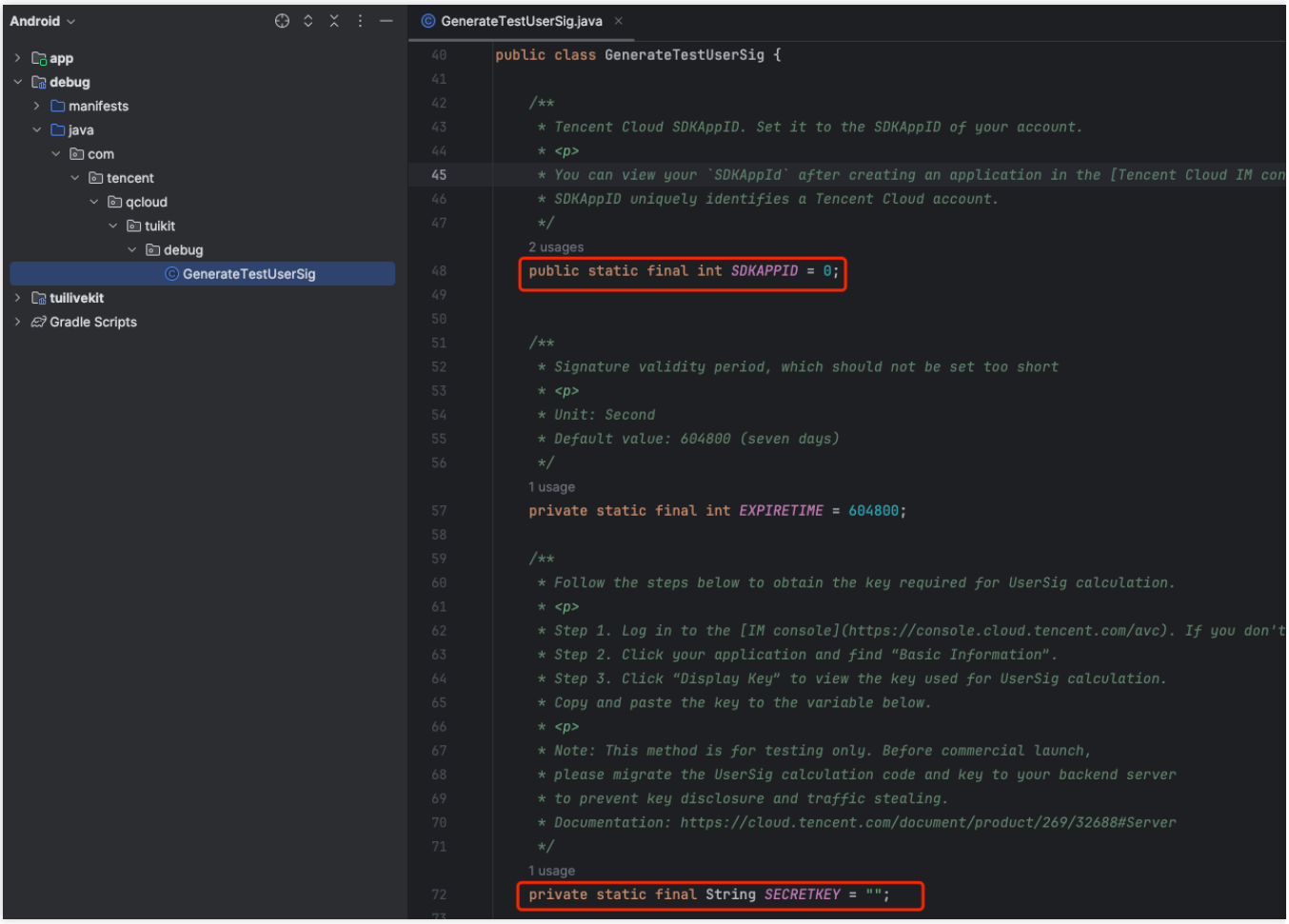


Step 2: Configure the Demo

1. Activate the TRTC service, obtain the SDKAppID and SDKSecretKey.

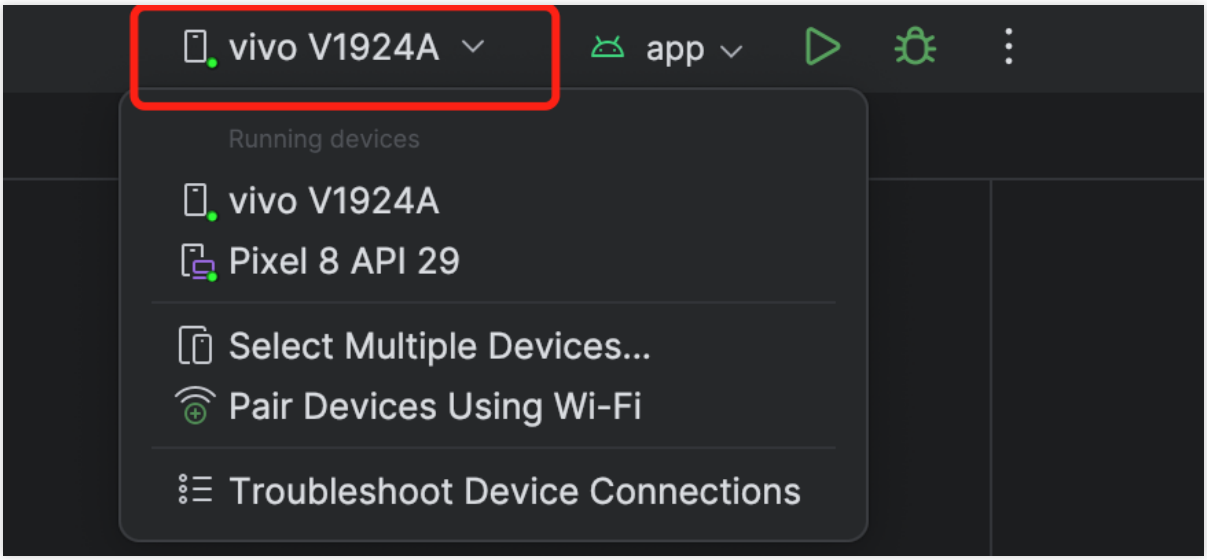


2. Open the Android/debug/src/main/java/com/tencent/qcloud/tuikit/debug/GenerateTestUserSig.java file, and enter the SDKAppID and SDKSecretKey obtained when Activate the service:

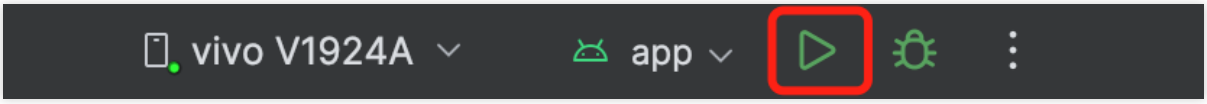


Step 3: Running the Demo

1. In the top right corner of Android Studio, select the device you want to run the Demo on as shown below:



2. After selecting, click **Run** to execute the TUILiveKit Android Demo on the target device.



3. After the demo is successfully run on the device, you can start and watch live broadcasts by following the steps below:

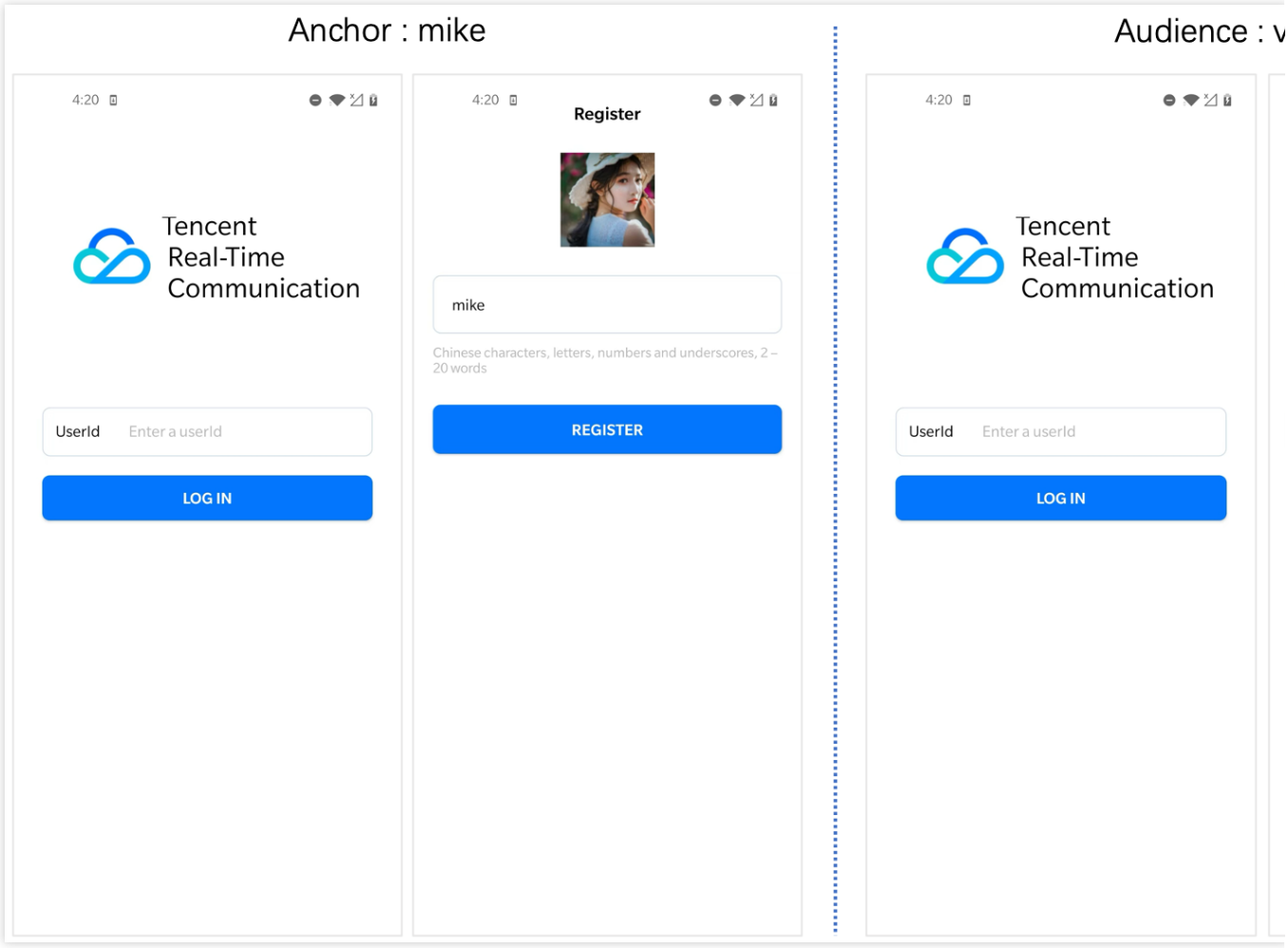
Start Live Broadcast

Note：

In order to allow you to experience the complete video live broadcast process, please log in two users on two devices to use the Demo, one as the host and the other as the audience.

1. Log in & Signup

Please enter your UserId in the User ID field. If your current User ID has not been used before, you will be taken to the Registration page where you can set an avatar and nickname for yourself.



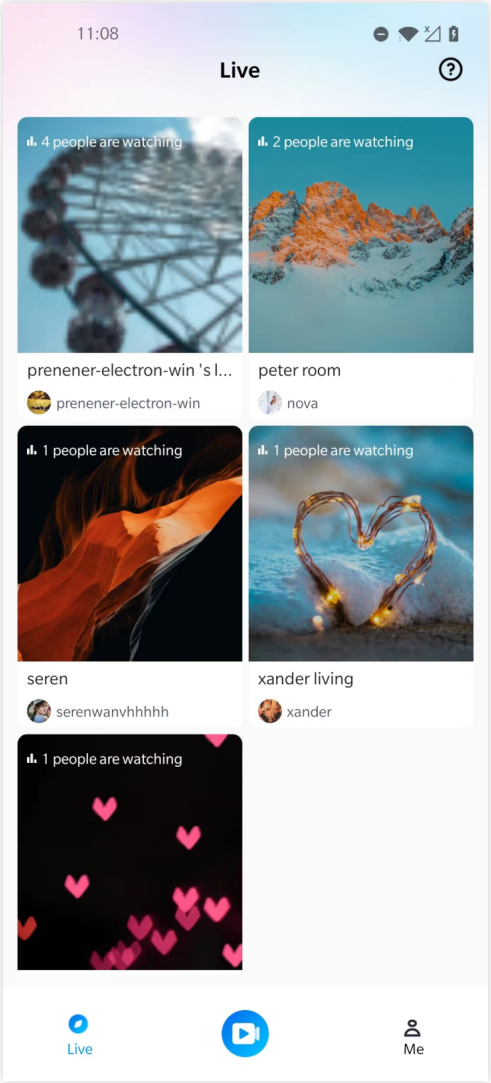
Note：

Try to avoid setting your User ID to simple strings like "1", "123", "111", as TRTC does not support the same User ID being logged into from multiple devices. Such User IDs like "1", "123", "111" are easily occupied by your colleagues during collaborative development, leading to login failures. Therefore, we recommend setting highly recognizable User IDs while debugging.

2. The anchor starts the live broadcast.

Click the **Start Broadcast** button in the middle of the bottom of the homepage to enter the broadcast preview page, and then click **Start Live** to start the live broadcast.

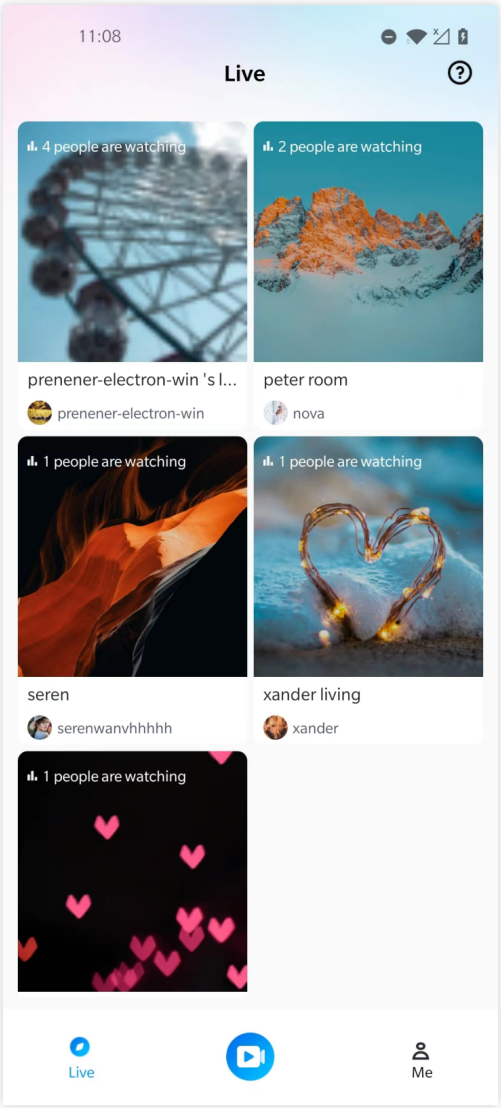
Anchor: Before entering	Anchor: Preview	

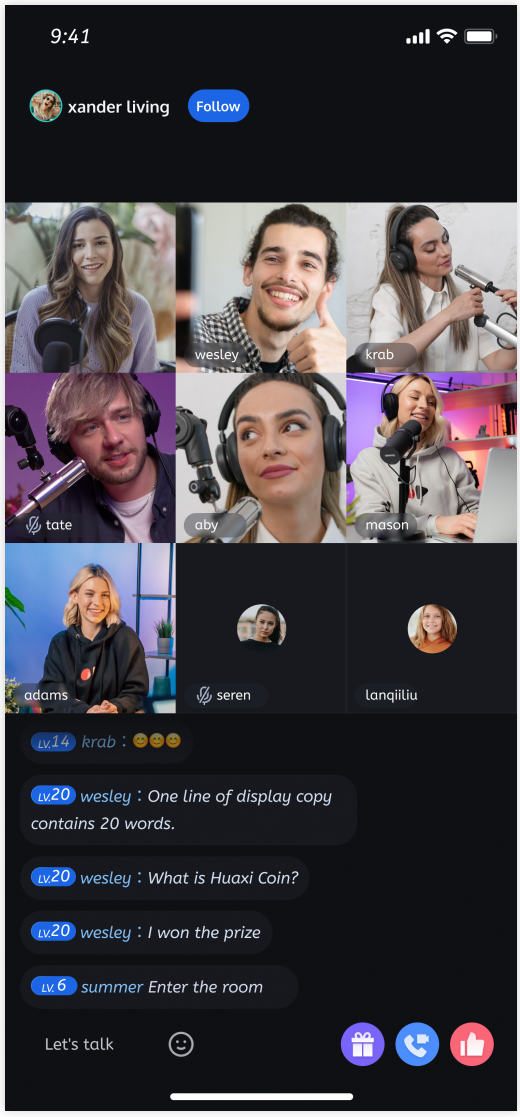


3. Viewers join the live broadcast room.

Click on any room in the live broadcast list to enter the live broadcast room.

Audience: Before entering	Audience: After entering

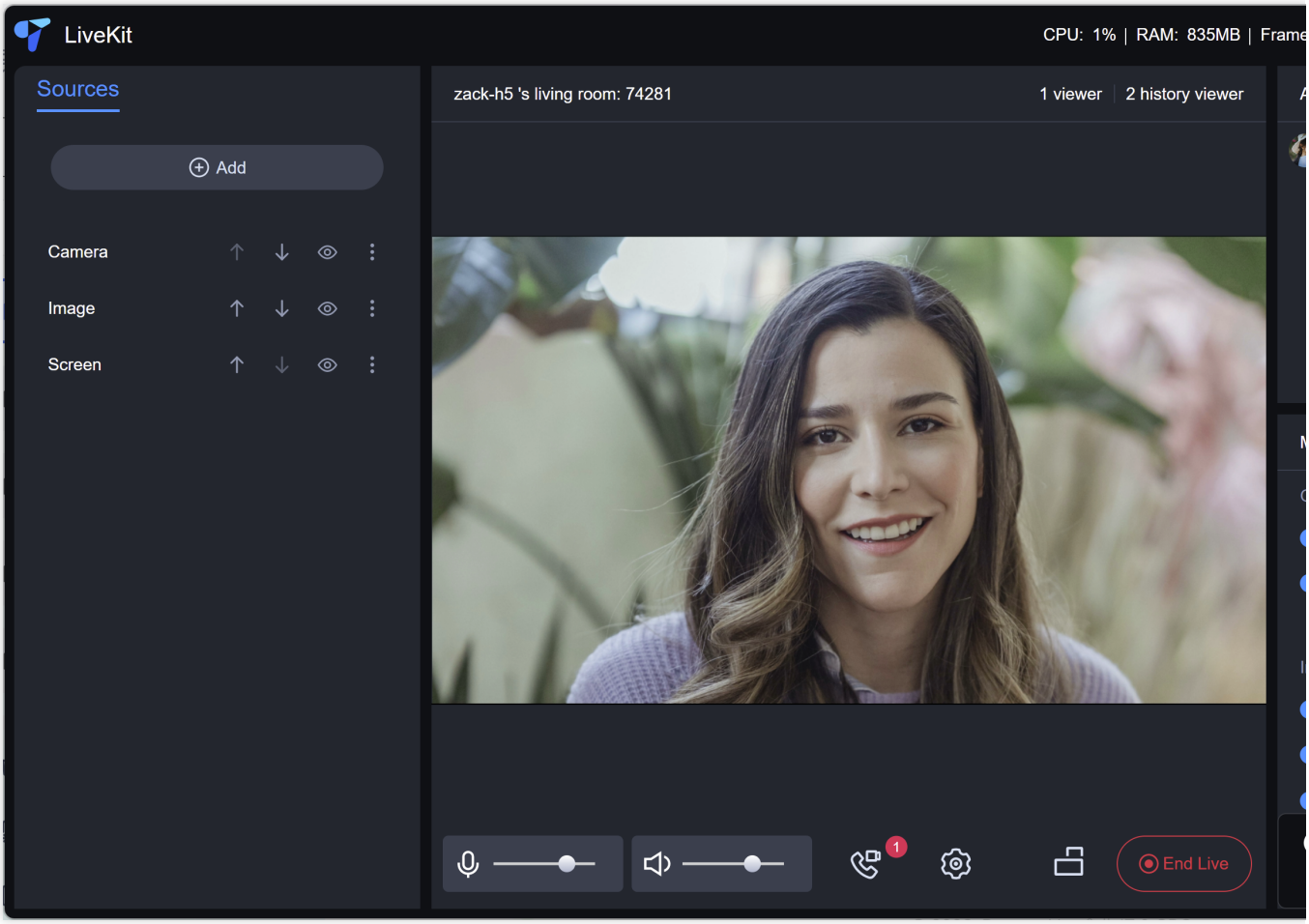




Electron

Last updated：2024-07-30 14:12:47

This article will introduce you to how to quickly build a **high-definition video live streaming application** up and run on the desktop. Following this document, you will start a live broadcast in 10 minutes. Desktop **high-definition video live streaming** supports **merging various multimedia resources into one video stream**, and allows for the editing, compositing, and custom layout of multimedia sources.

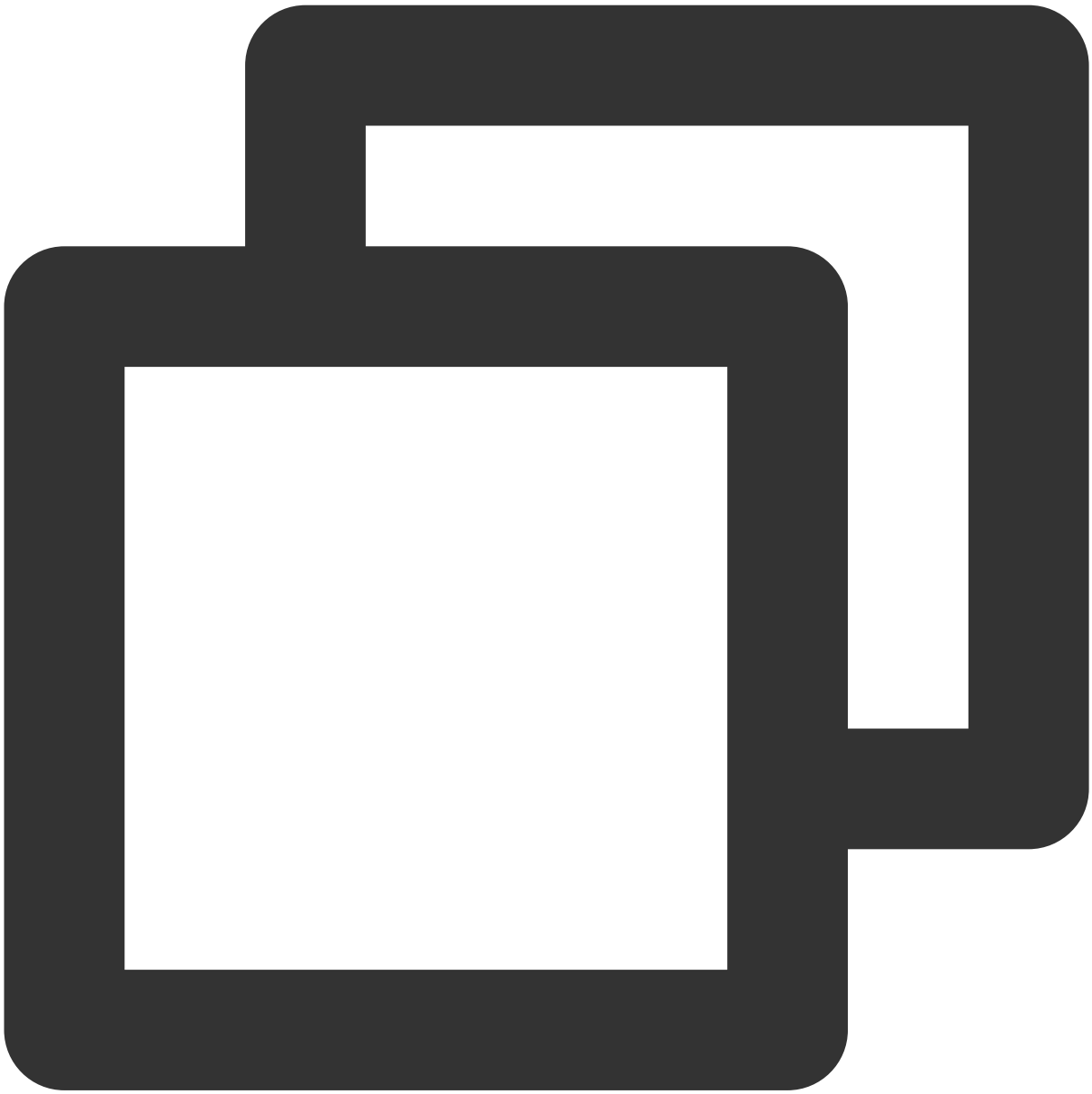


Environmental Preparation

Operating System: Windows 10 or 11.
Equipment Requirements: Camera, Microphone, Speaker.

Step 1: Download the Demo

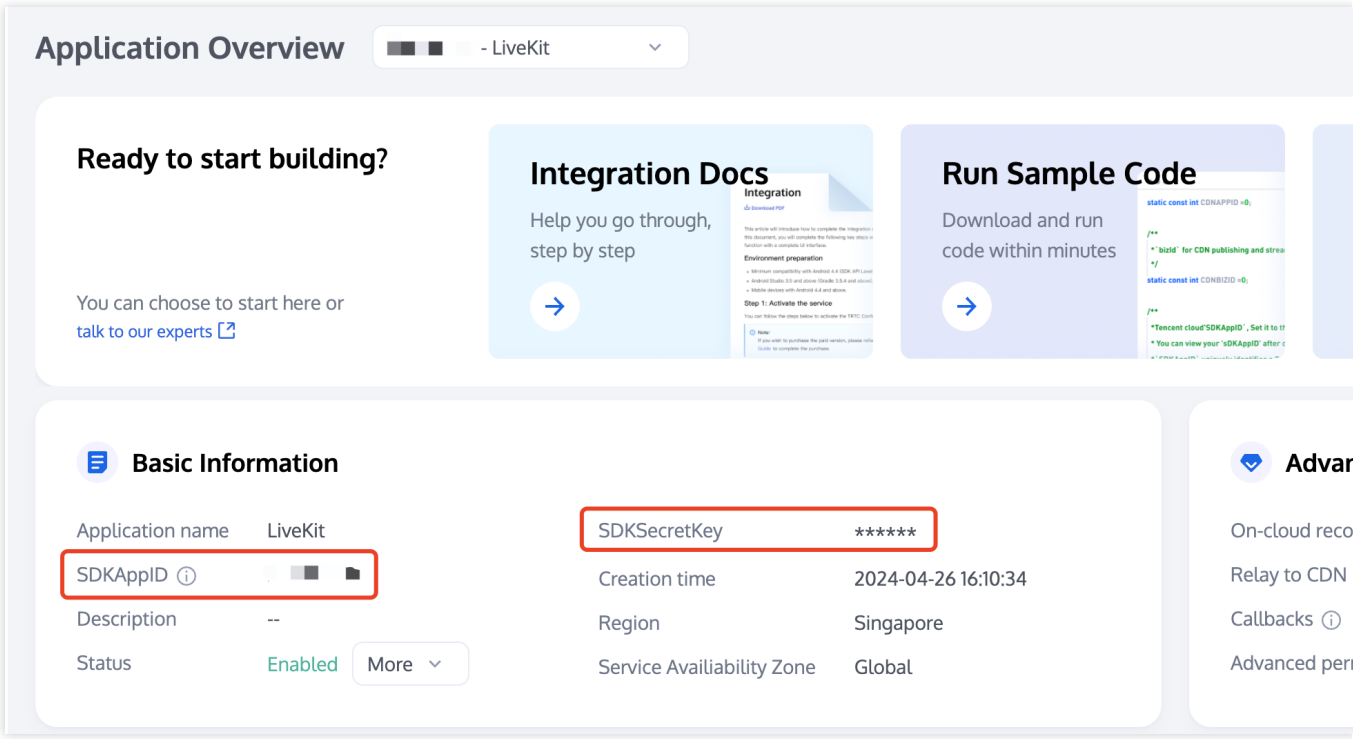
Get open source code from Github, or you can clone the code with the git command as below:



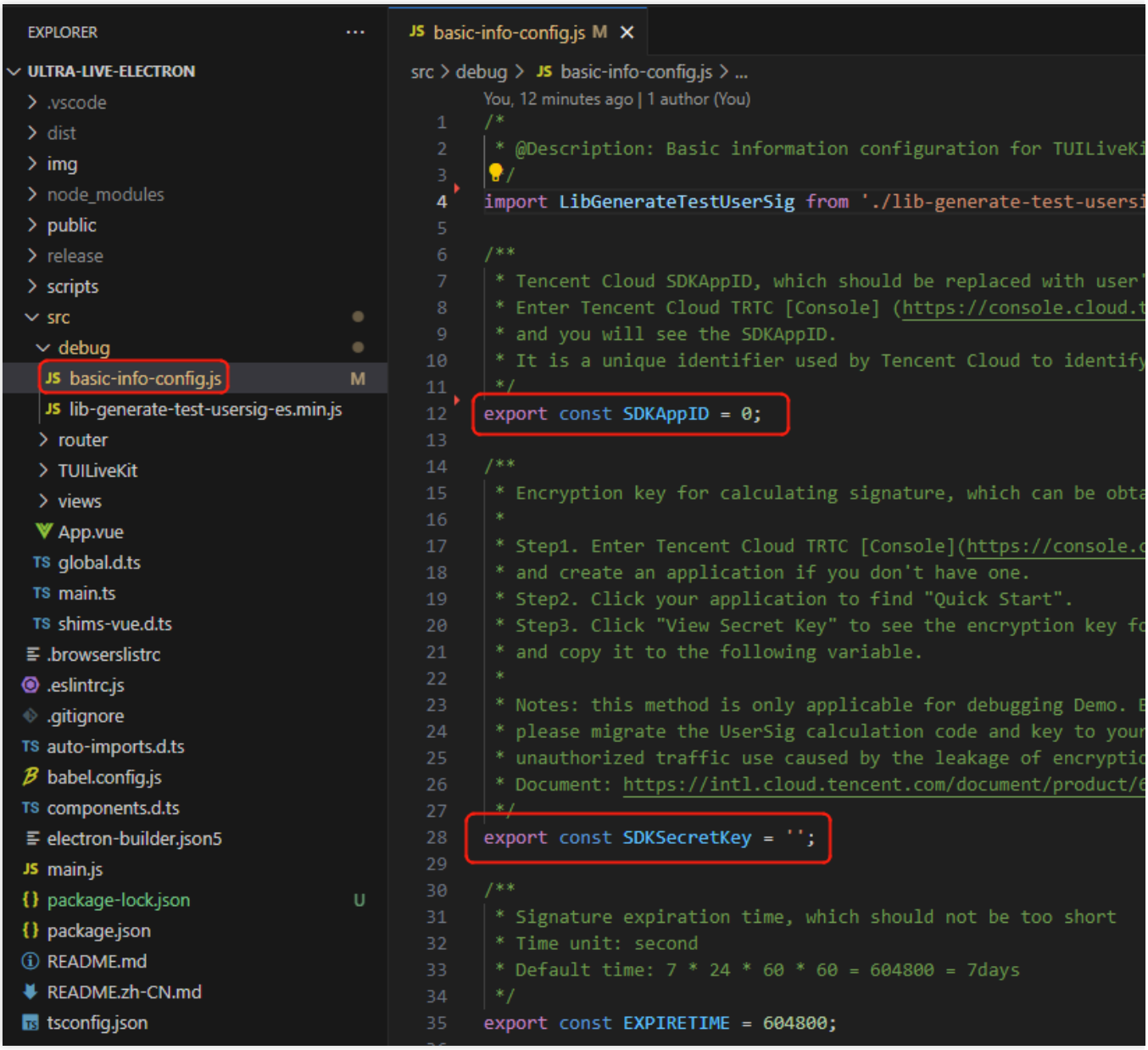
```
git clone https://github.com/Tencent-RTC/ultra-live-electron.git

cd ultra-live-electron
```

Step 2: Configuration

1. Activate the TRTC service, obtain the SDKAppID and SDKSecretKey.

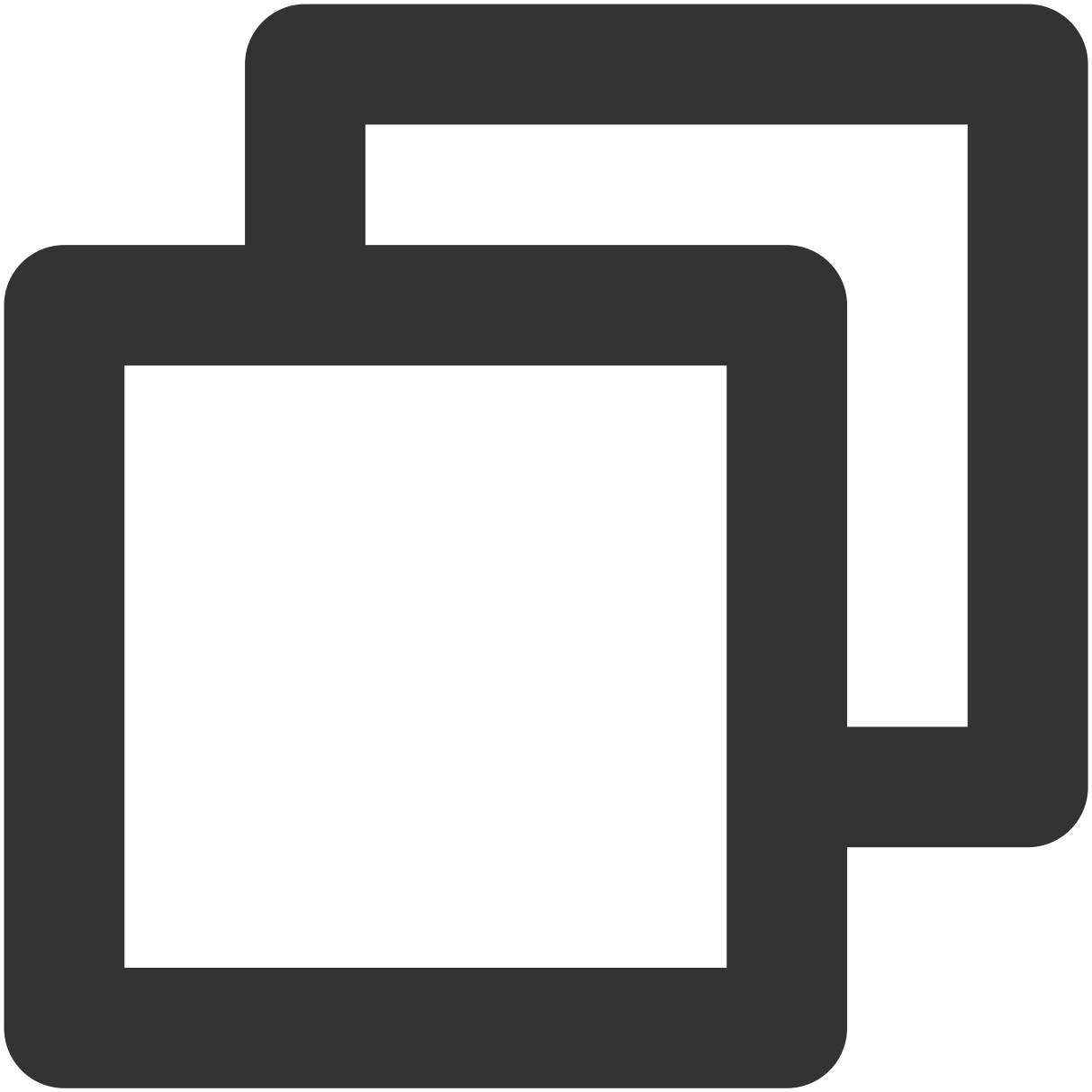


2. Open file `src/debug/basic-info-config.js` , and enter the SDKAppID and SDKSecretKey obtained when Activate the TRTC service:



Step 3: Run the demo

In the code directory, execute the following command to start the application in development mode:

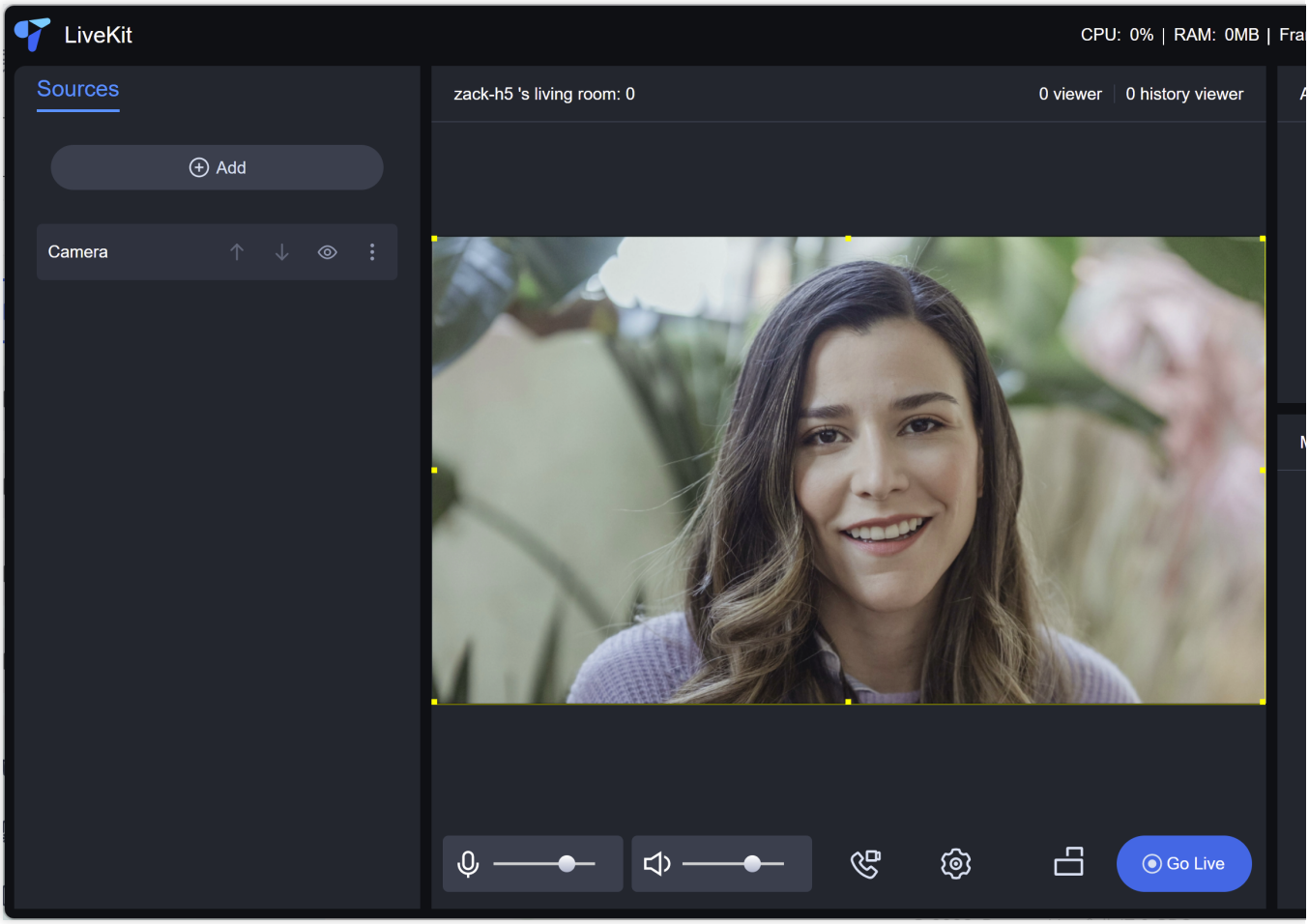


```
npm install

npm run start
```

Start your first live broadcast

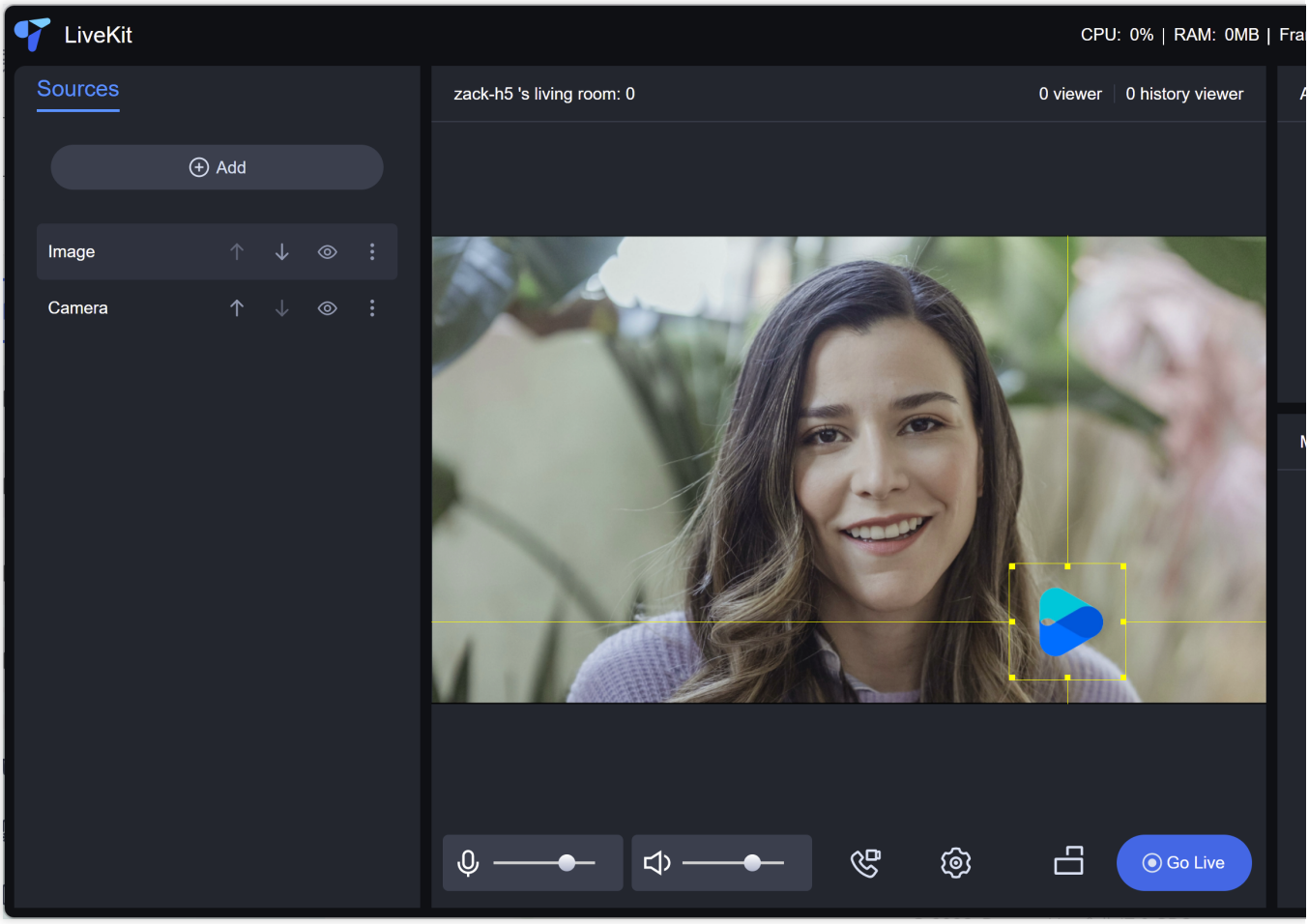
1. Add a camera

Firstly, you should add some multimedia source before start live broadcast. Multimedia sources supported include: camera, image, screen and window capture.For example, the image below shows the effect after adding a camera.



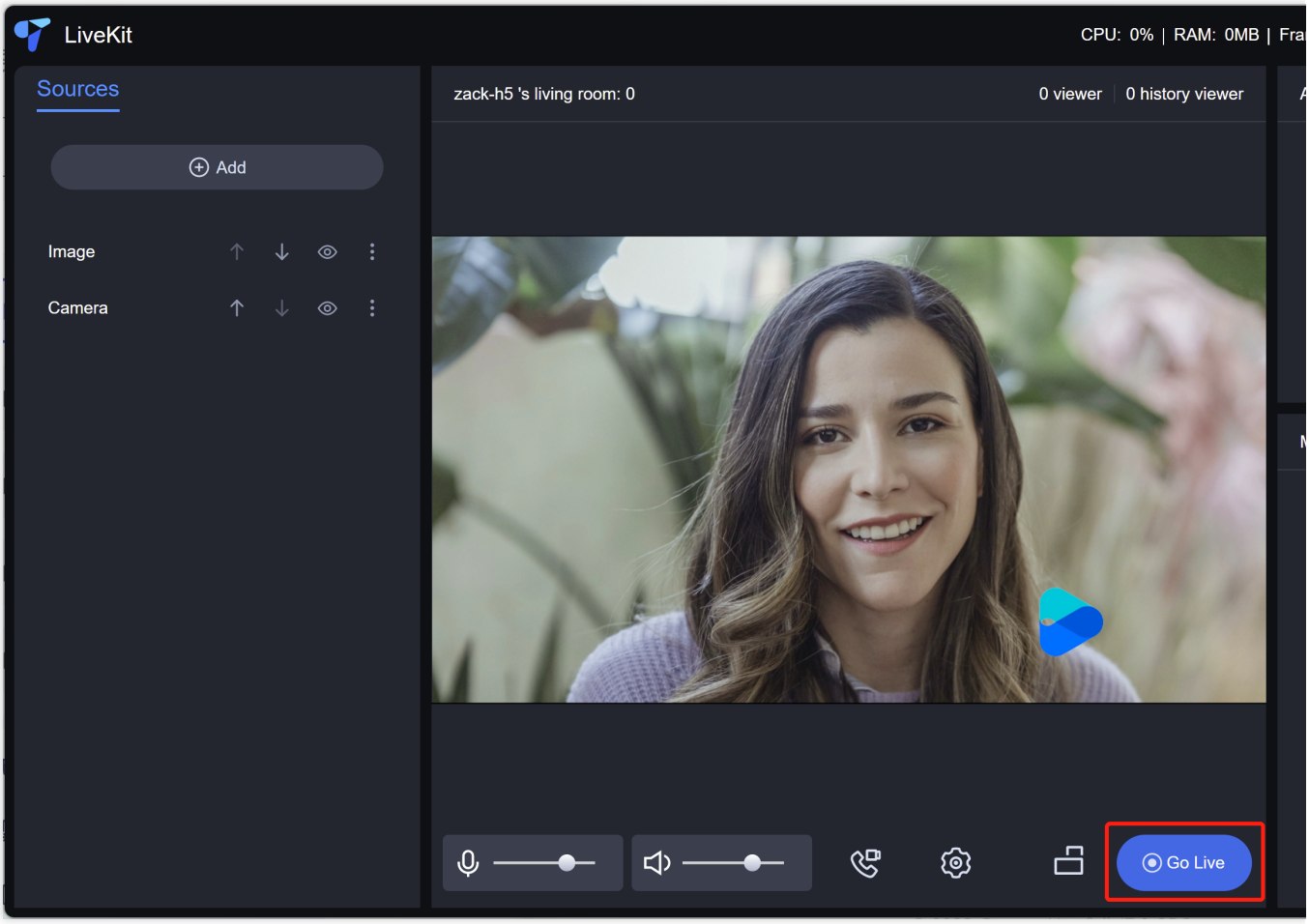
2. Add your logo image

If you need to add your own brand logo during a live broadcast, you can add a logo image. As shown in the image below, this is the effect after adding a transparent background logo image. The newly added image will have a yellow border around it, indicating that it is currently selected. A selected multimedia source can be moved and resized with mouse. It can also be rotated and modify its display level by right-click menu.



3. Start a live broadcast

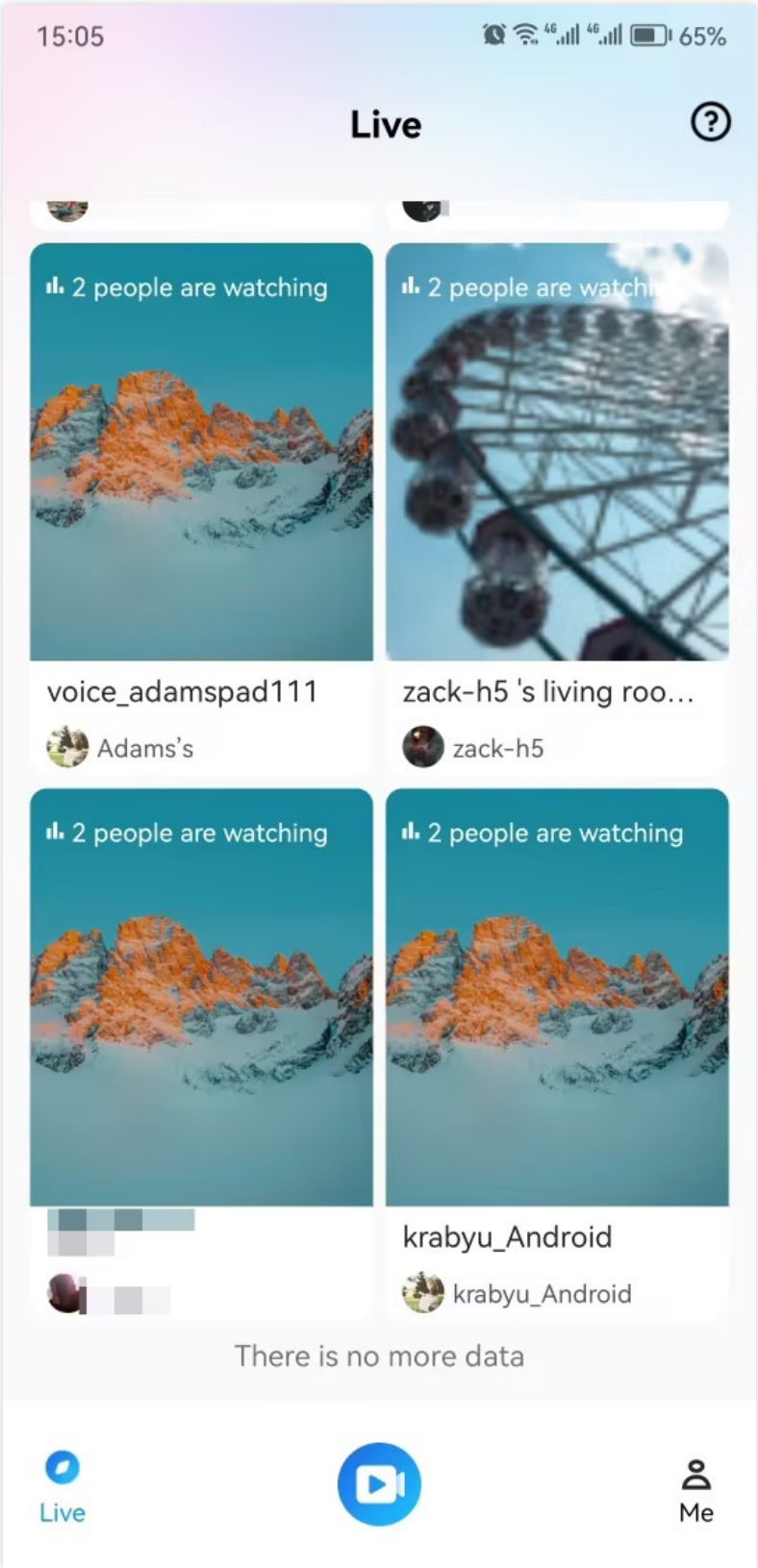
Click 'Go Live' button to start a live broadcast . Once the broadcast starts successfully, the 'Go Live' button will turn into 'End'. Click it to end the live boradcast.

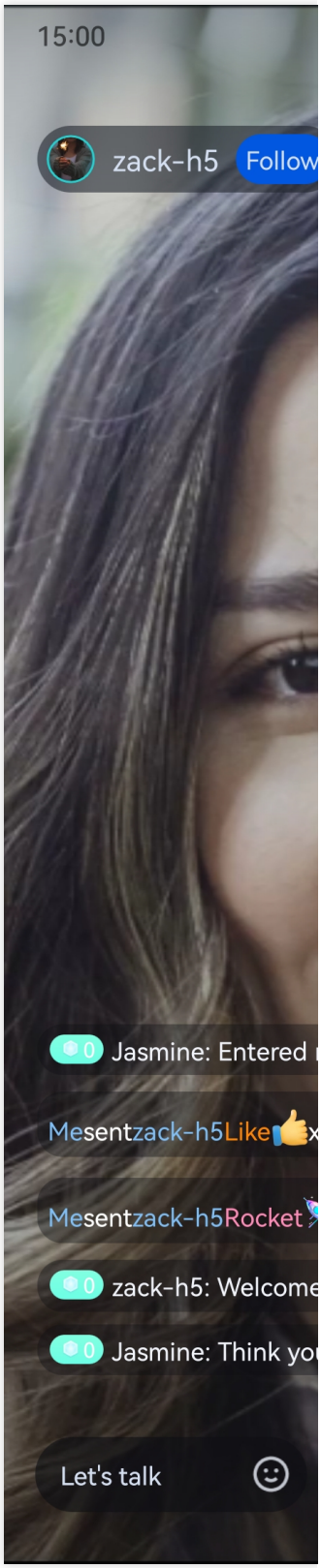


4. View the live broadcast

The desktop version only supports the host starting the broadcast. To watch, you need to use the mobile app. Find the corresponding live room in the live list on the mobile app and enter the live room. For the use of the mobile app, please refer to the documentation for iOS and Android.

Audience: explore the live list	Audience: Enter a live room



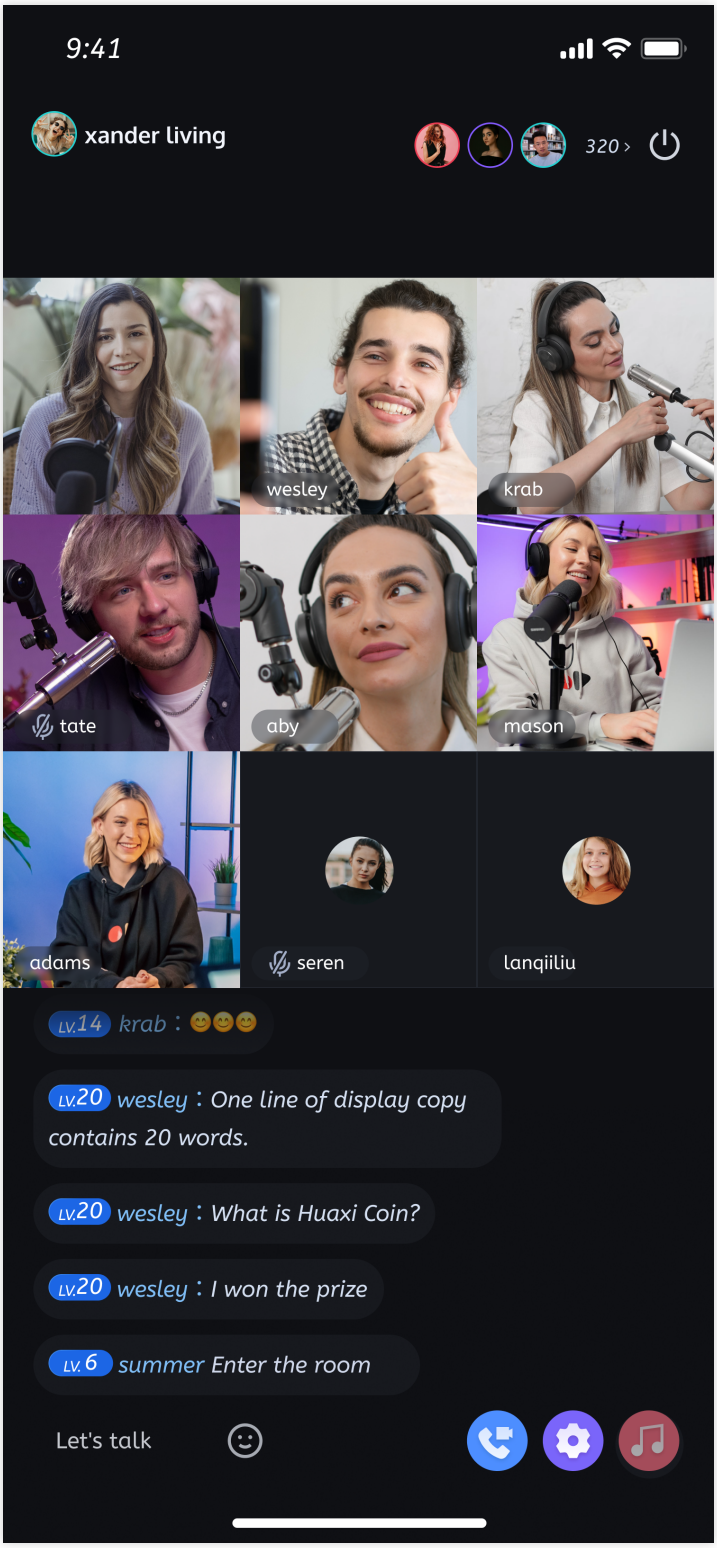


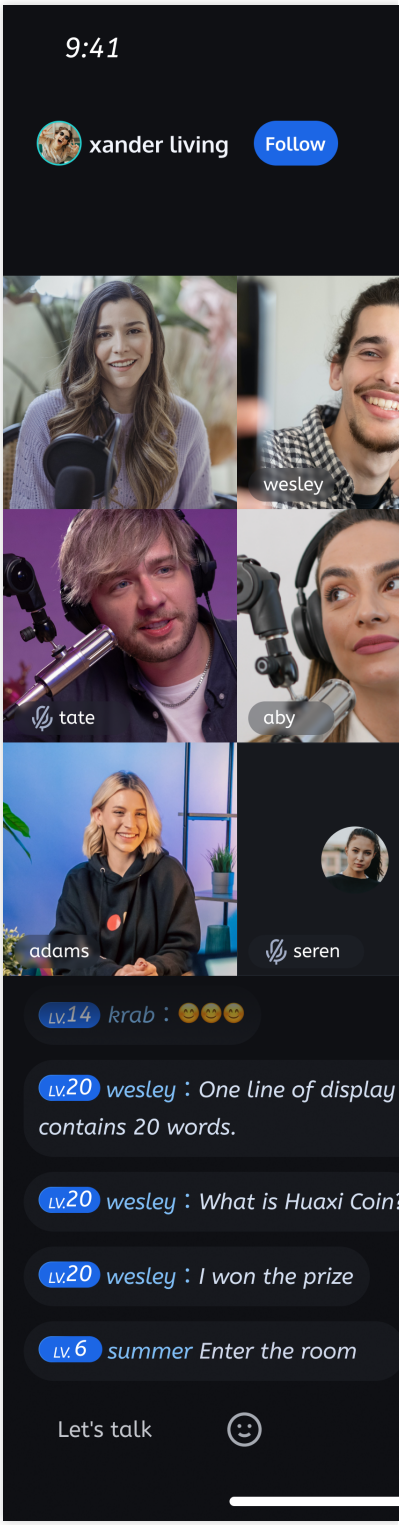
Flutter

Last updated：2024-08-13 17:40:11

This article will show you how to quickly run the video live streaming demo. Following this document, you can run the demo in 10 minutes and finally experience a video live streaming function with a complete UI interface.

Anchor	Audience





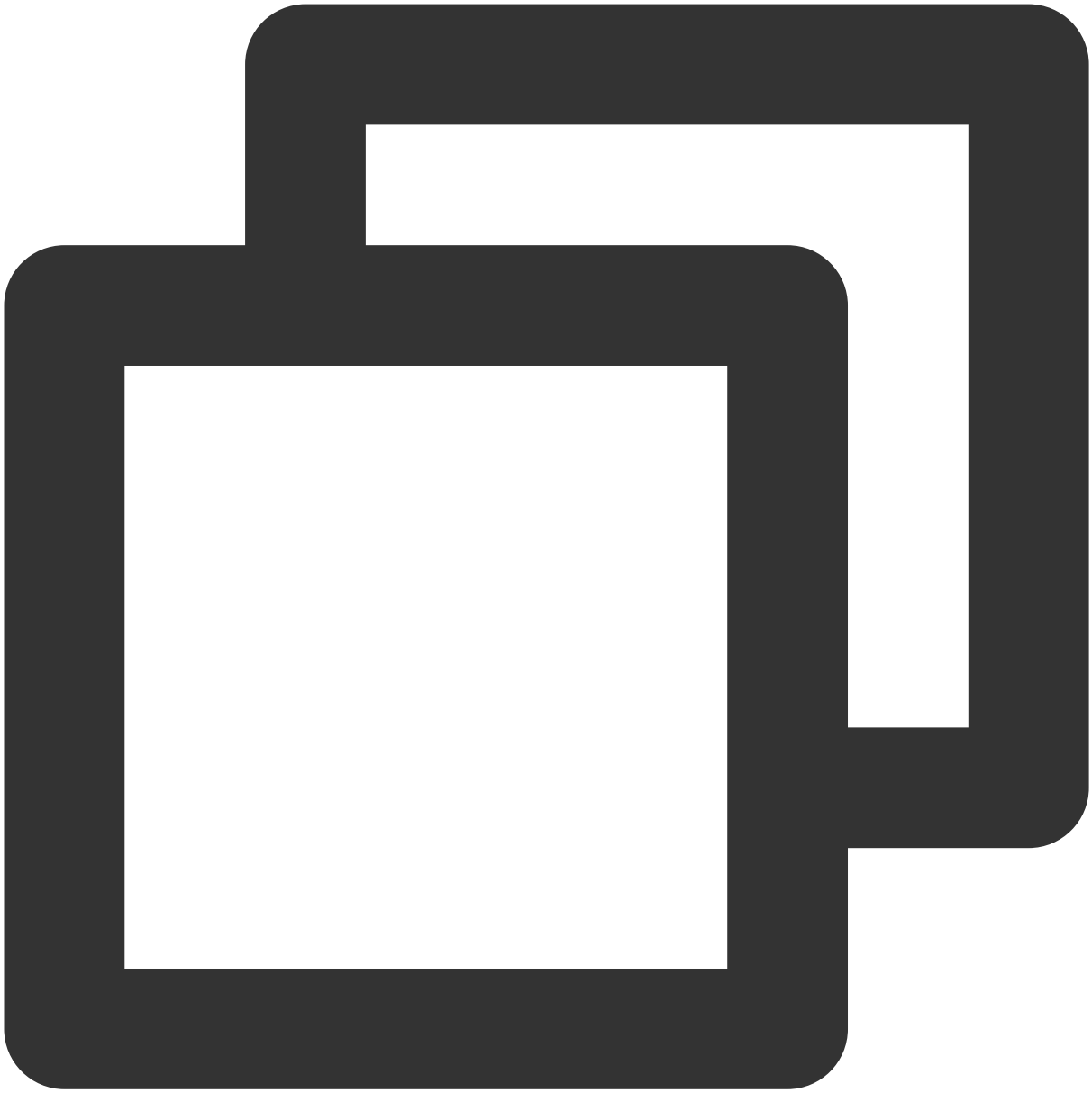
Environment Preparations

Platform	Version

Flutter	Flutter 3.22.0 or later.。 Dart version 3.4.0 or higher.
Android	Android Studio 3.5 or later. Android devices 5.0 or later.
iOS	Xcode 13.0 or later. Please ensure that your project has a valid developer signature set.

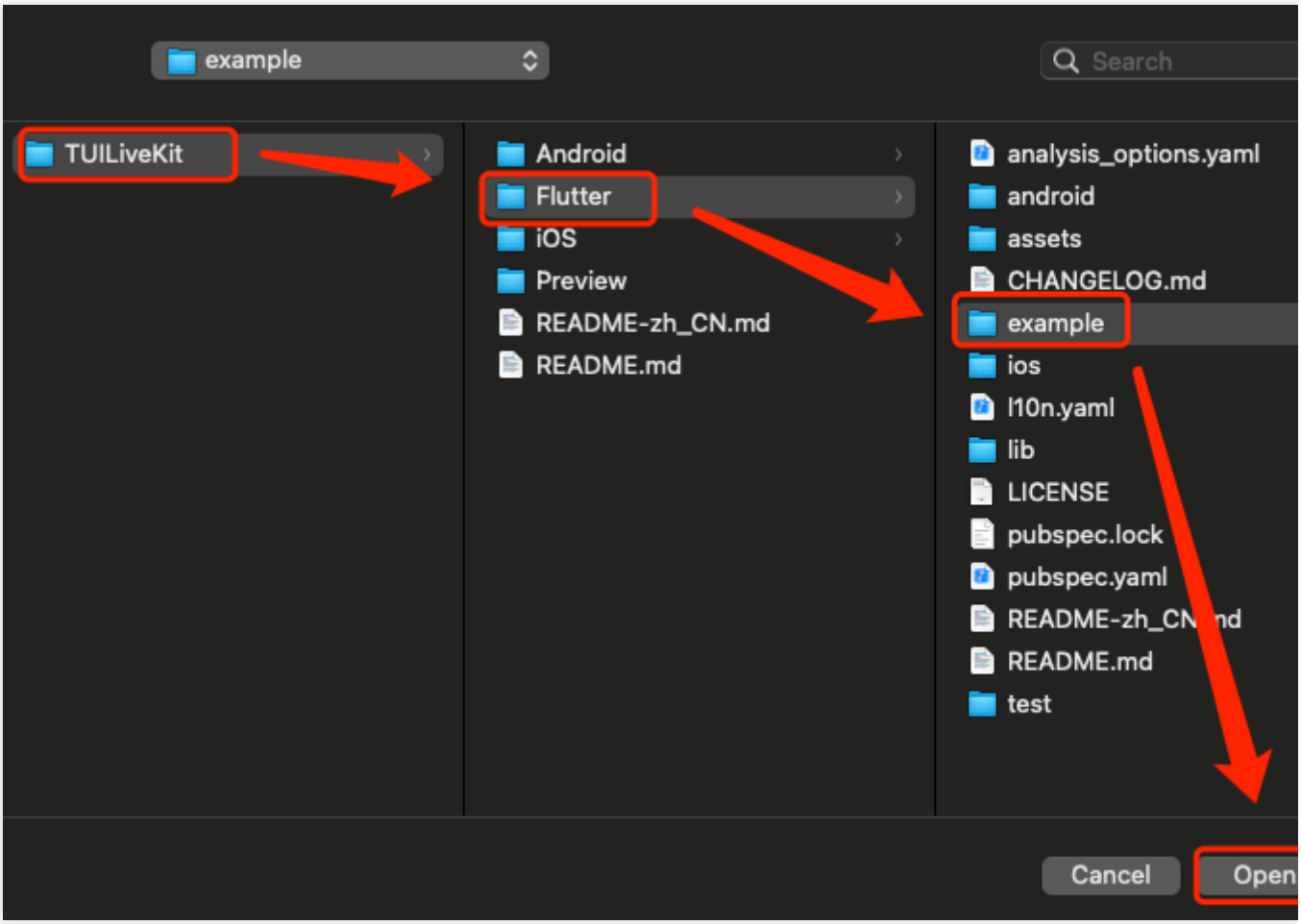
Step 1: Download the Demo

1. Download the TUILiveKit Demo source code from GitHub, or directly run the following command in the command line:



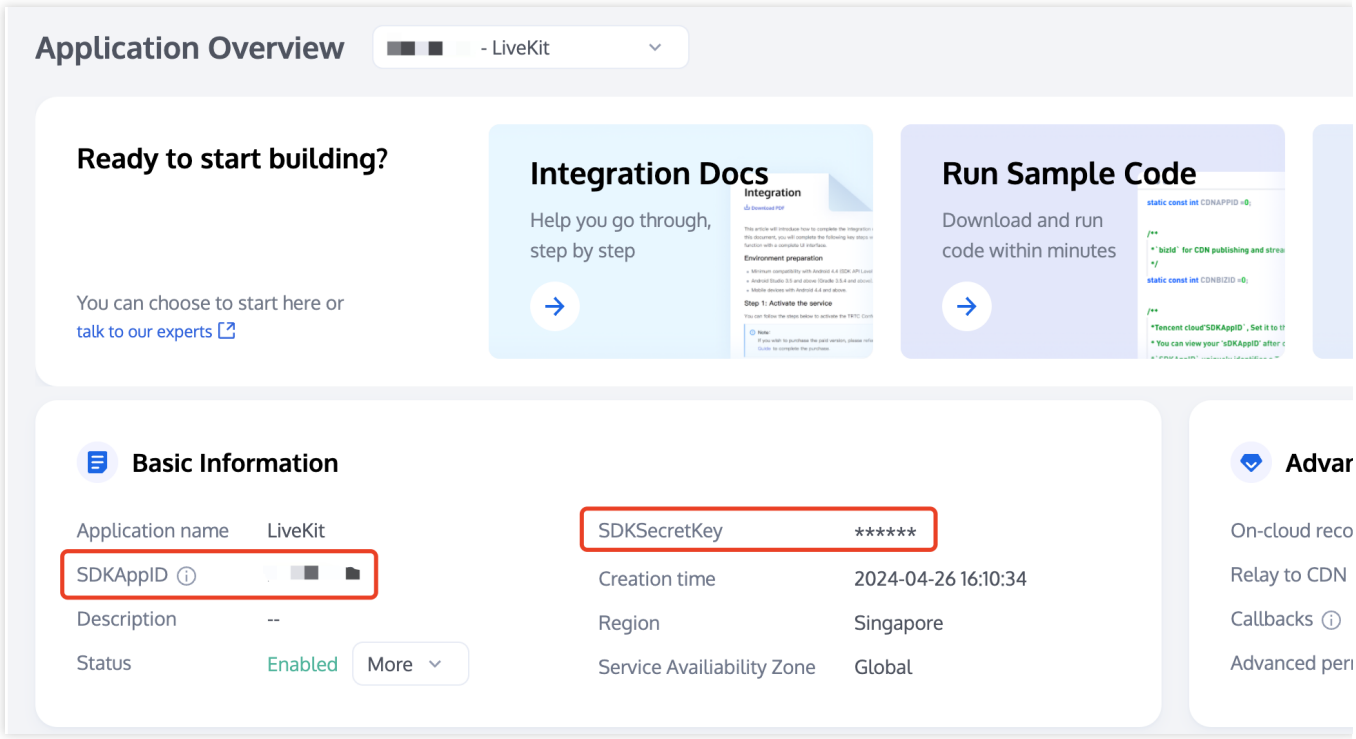
```
git clone https://github.com/Tencent-RTC/TUILiveKit.git
```

2. Open the TUILiveKit **Flutter** project through Android Studio:

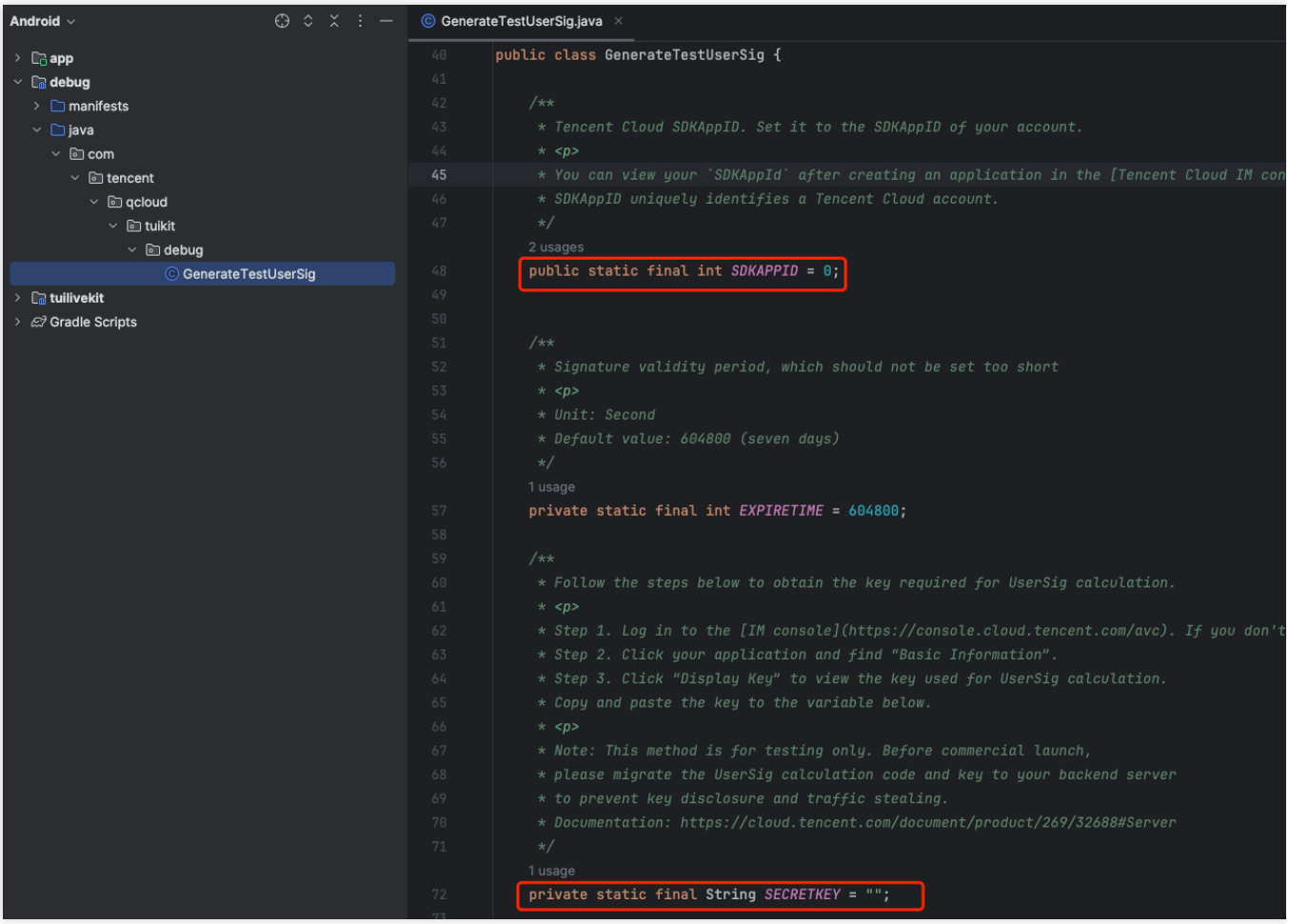


Step 2: Configure the Demo

1. Activate the TRTC service, obtain the SDKAppID and SDKSecretKey.

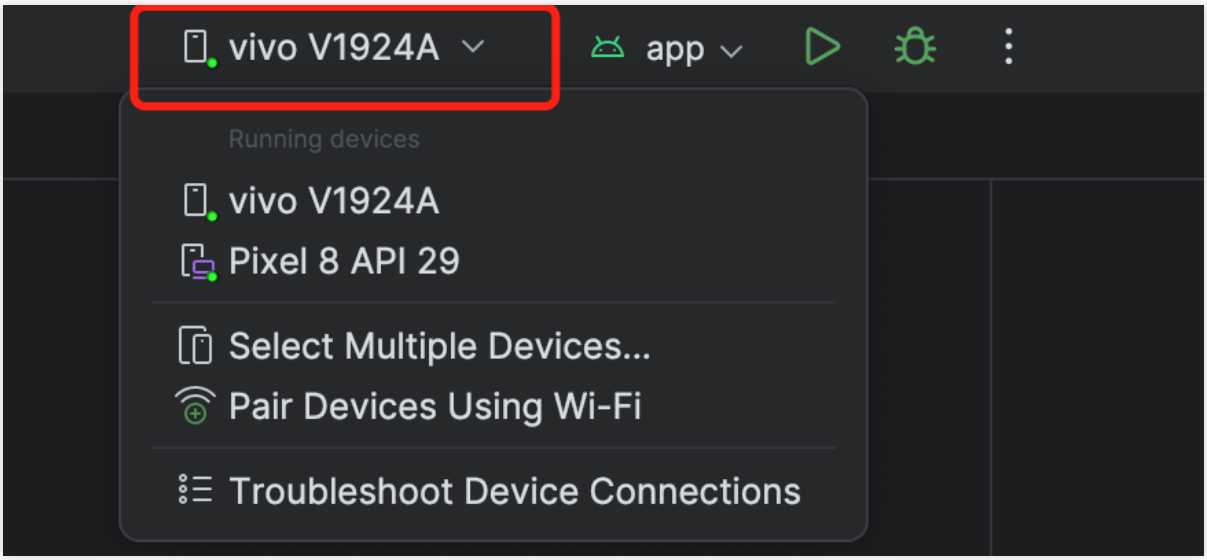


2. Open the Flutter/example/lib/debug/generate_test_user_sig.dart file and fill in the corresponding SDKAppID and SDKSecretKey obtained when Activate the service:

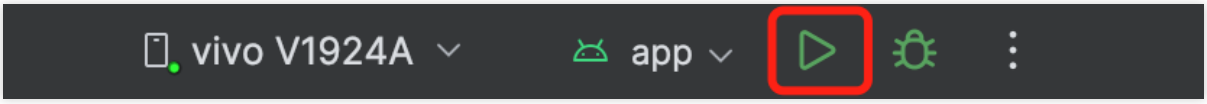


Step 3: Running the Demo

1. In the top right corner of Android Studio, select the device you want to run the Demo on as shown below:



2. After selecting, click **Run** to execute the TUILiveKit Flutter Demo on the target device.



3. After the demo is successfully run on the device, you can start and watch live broadcasts by following the steps below:

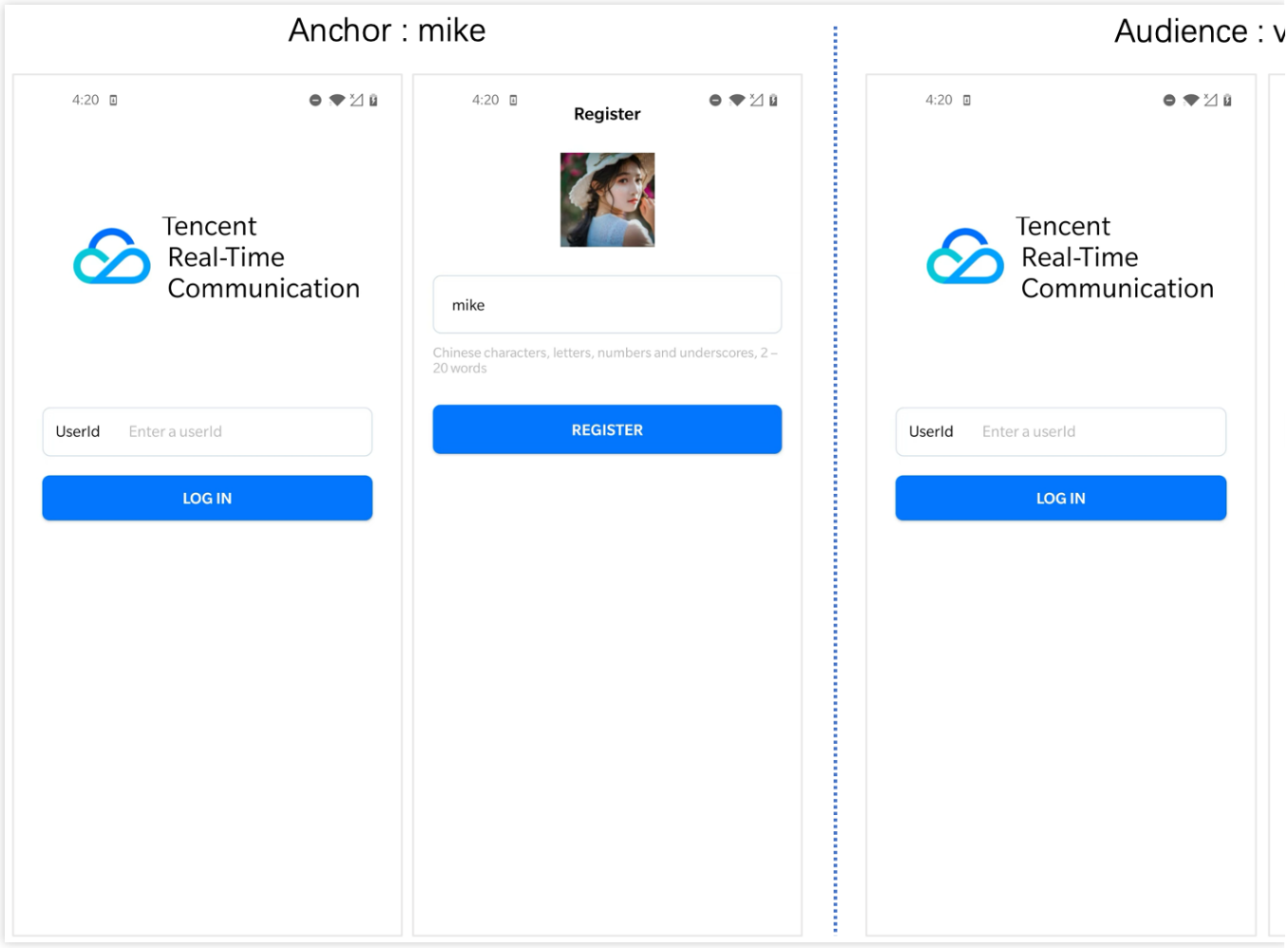
Start Live Broadcast

Note：

In order to allow you to experience the complete video live broadcast process, please log in two users on two devices to use the Demo, one as the host and the other as the audience.

1. Log in & Signup

Please enter your UserId in the User ID field. If your current User ID has not been used before, you will be taken to the Registration page where you can set an avatar and nickname for yourself.



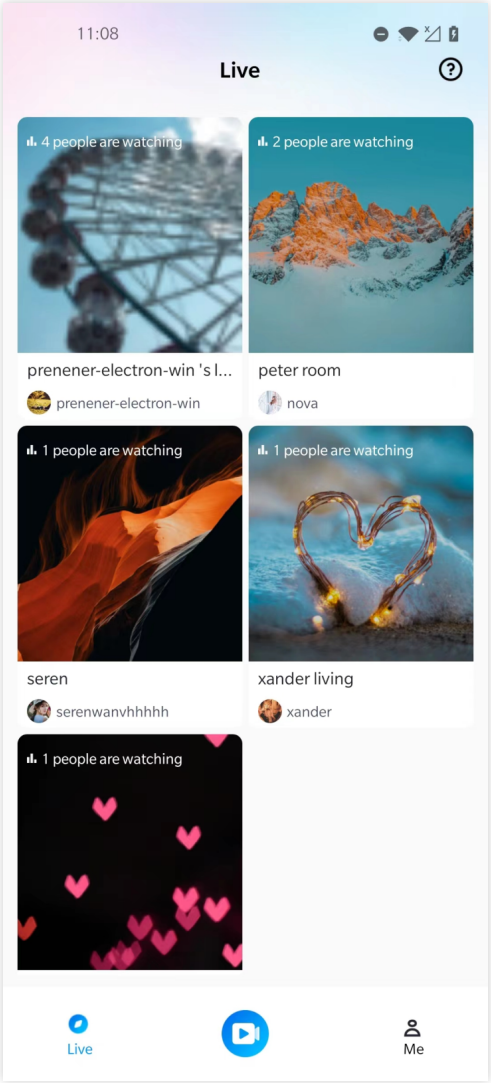
Note：

Try to avoid setting your User ID to simple strings like "1", "123", "111", as TRTC does not support the same User ID being logged into from multiple devices. Such User IDs like "1", "123", "111" are easily occupied by your colleagues during collaborative development, leading to login failures. Therefore, we recommend setting highly recognizable User IDs while debugging.

2. The anchor starts the live broadcast.

Click the **Start Broadcast** button in the middle of the bottom of the homepage to enter the broadcast preview page, and then click **Go Live** to start the live broadcast.

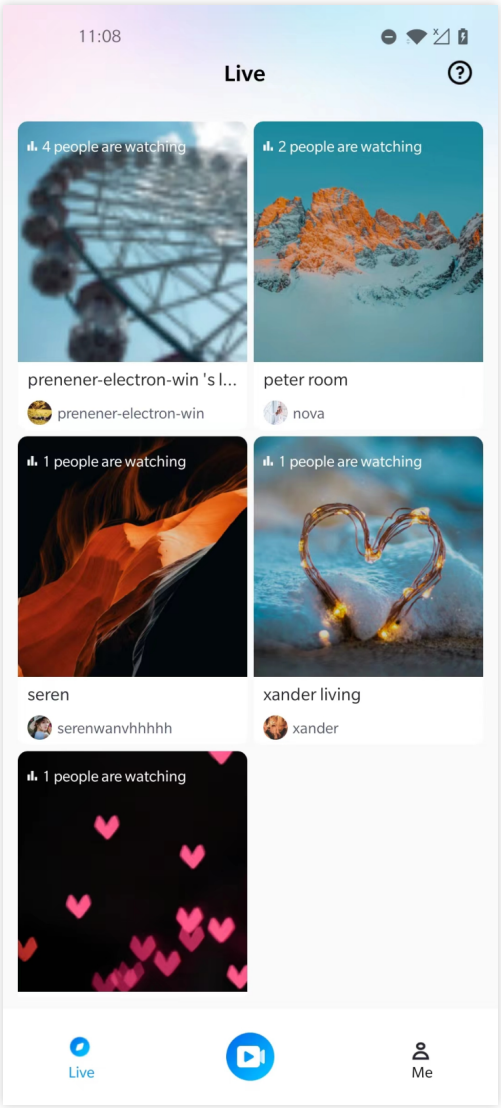
Anchor: Before entering	Anchor: Preview	

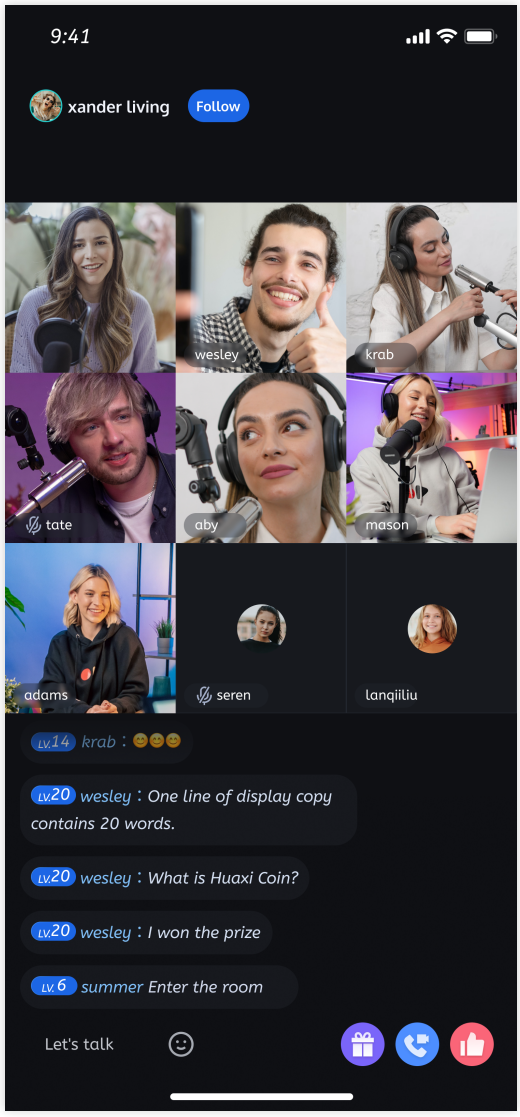


3. Viewers join the live broadcast room.

Click on any room in the live broadcast list to enter the live broadcast room.

Audience: Before entering	Audience: After entering





Integration (TUILiveKit)
iOS

Last updated：2024-08-09 22:25:01

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

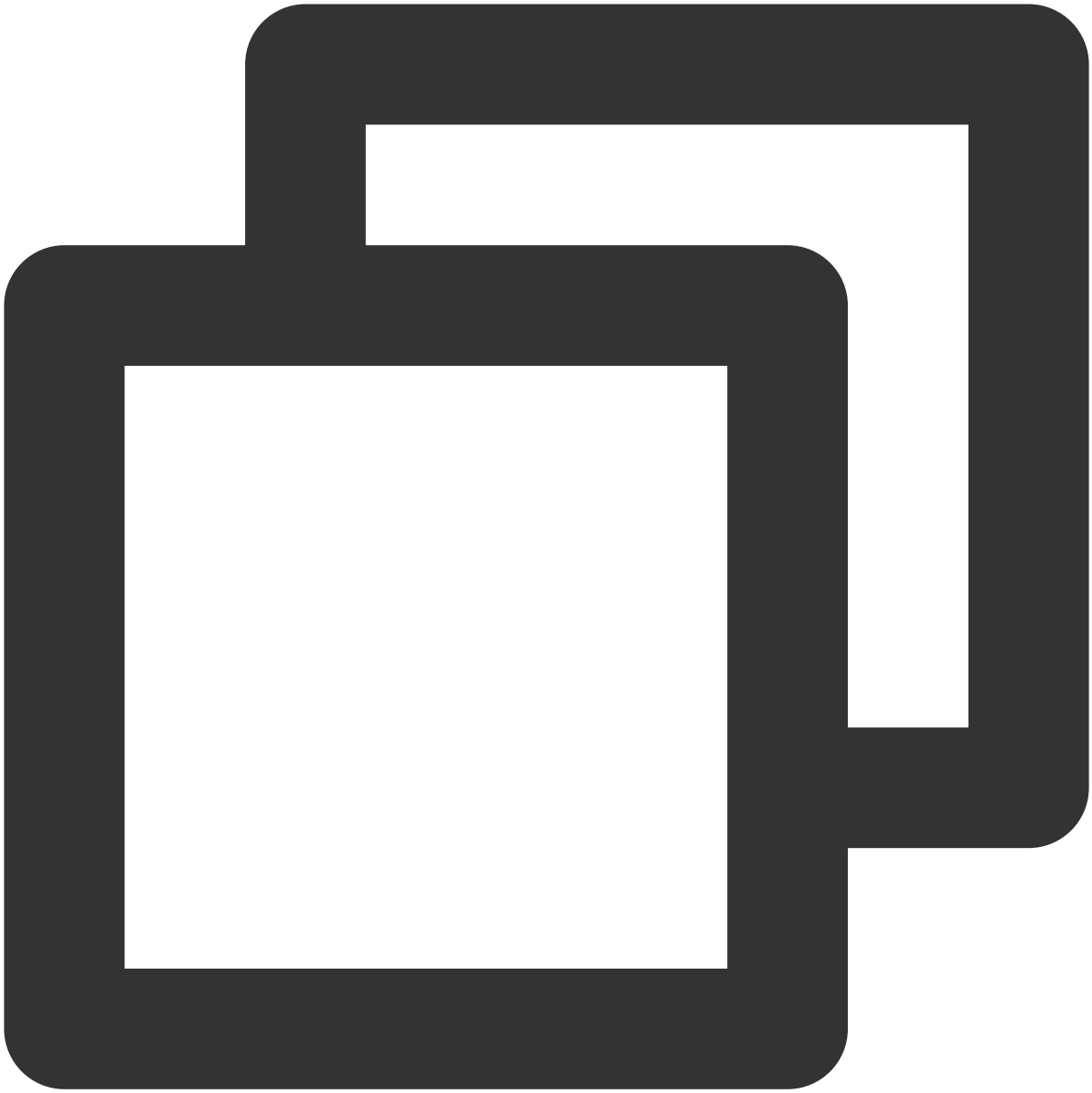
Xcode 15 or later
iOS 13.0 or later
CocoaPods environment installation, click to view.
If you encounter problems with access and use, see Q&A。

Step 1. Activate the service

Before using 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

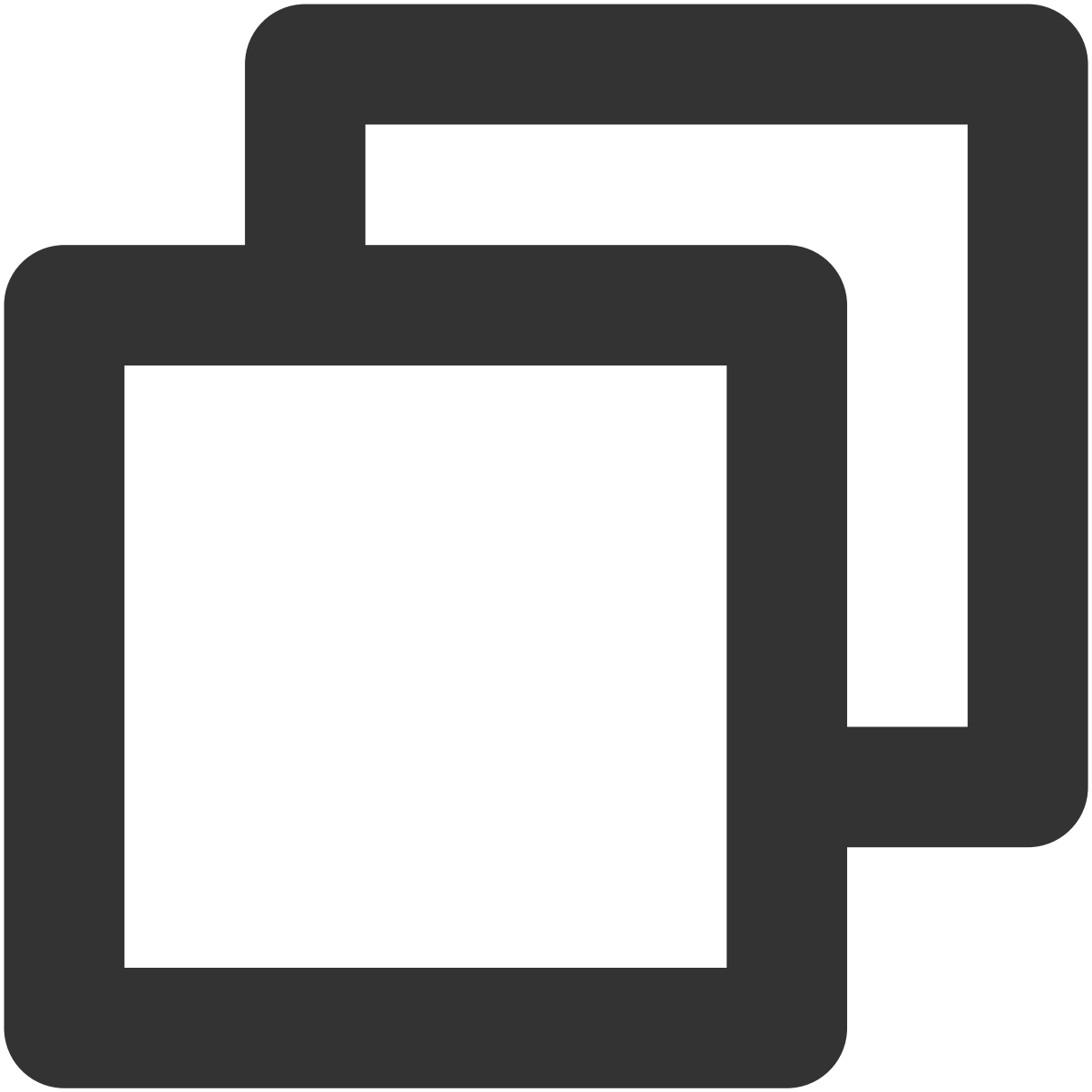
Step 2. Import the component

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.



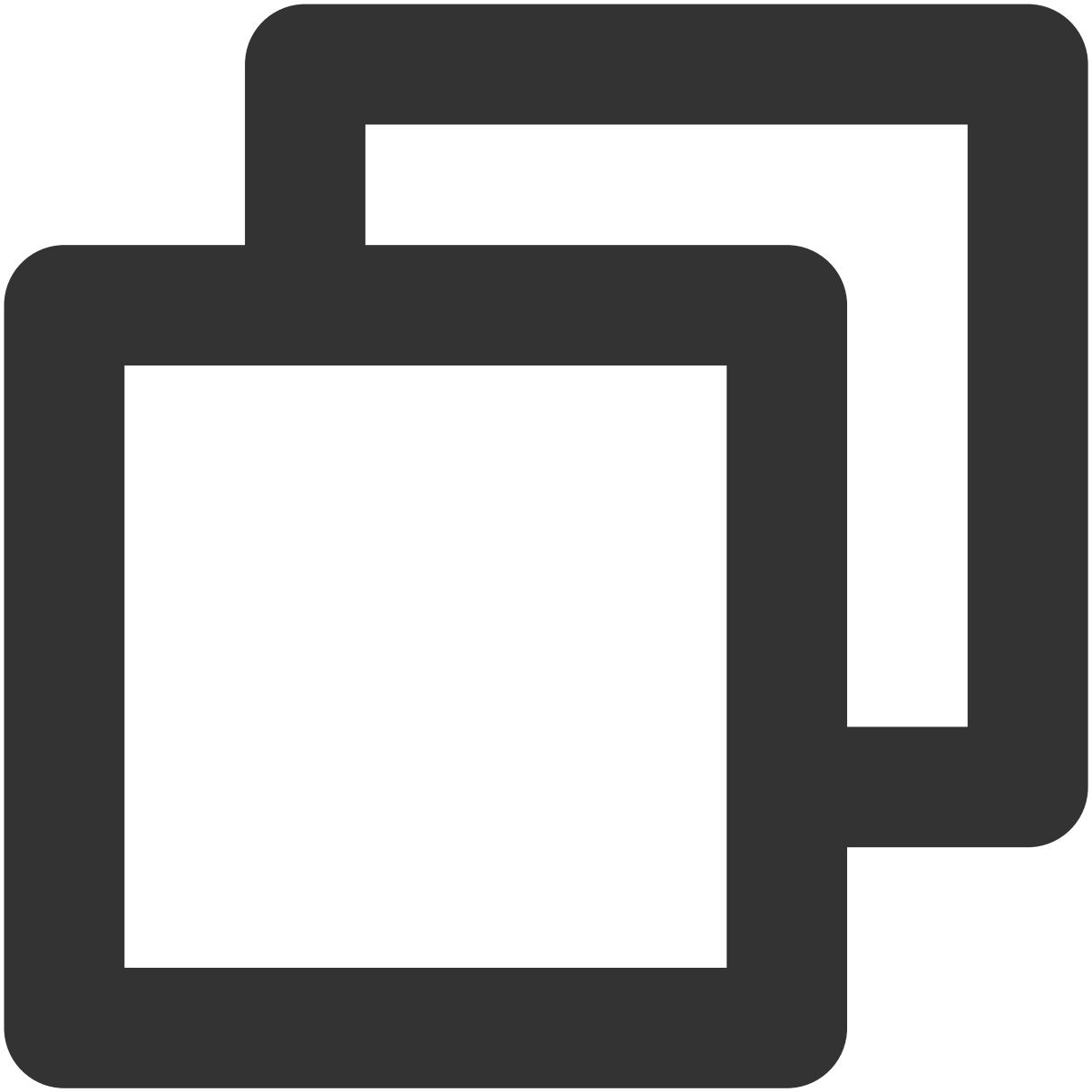
```
target 'xxxx' do
  ...
  ...
  pod 'TUILiveKit'
end
```

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



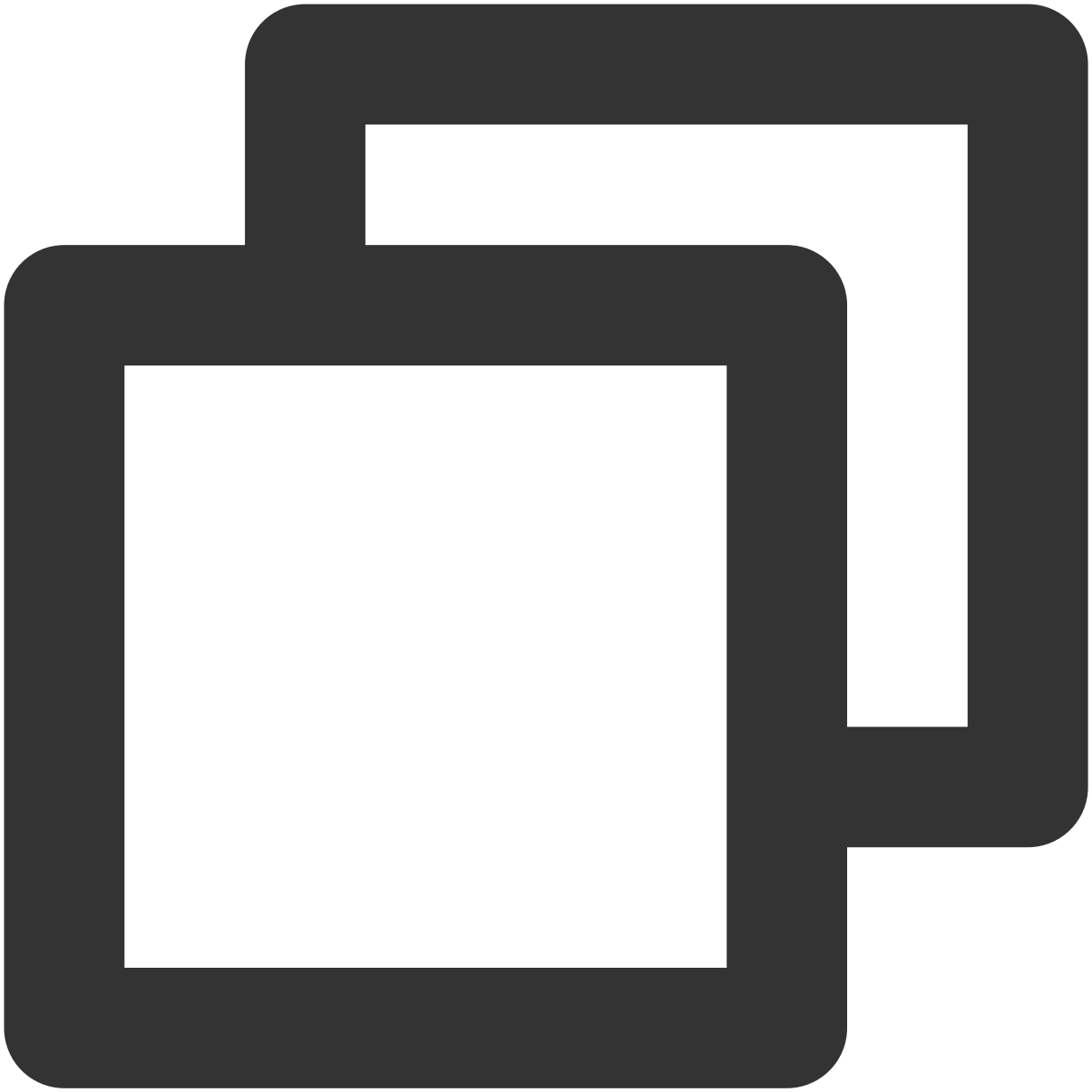
```
pod init
```

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



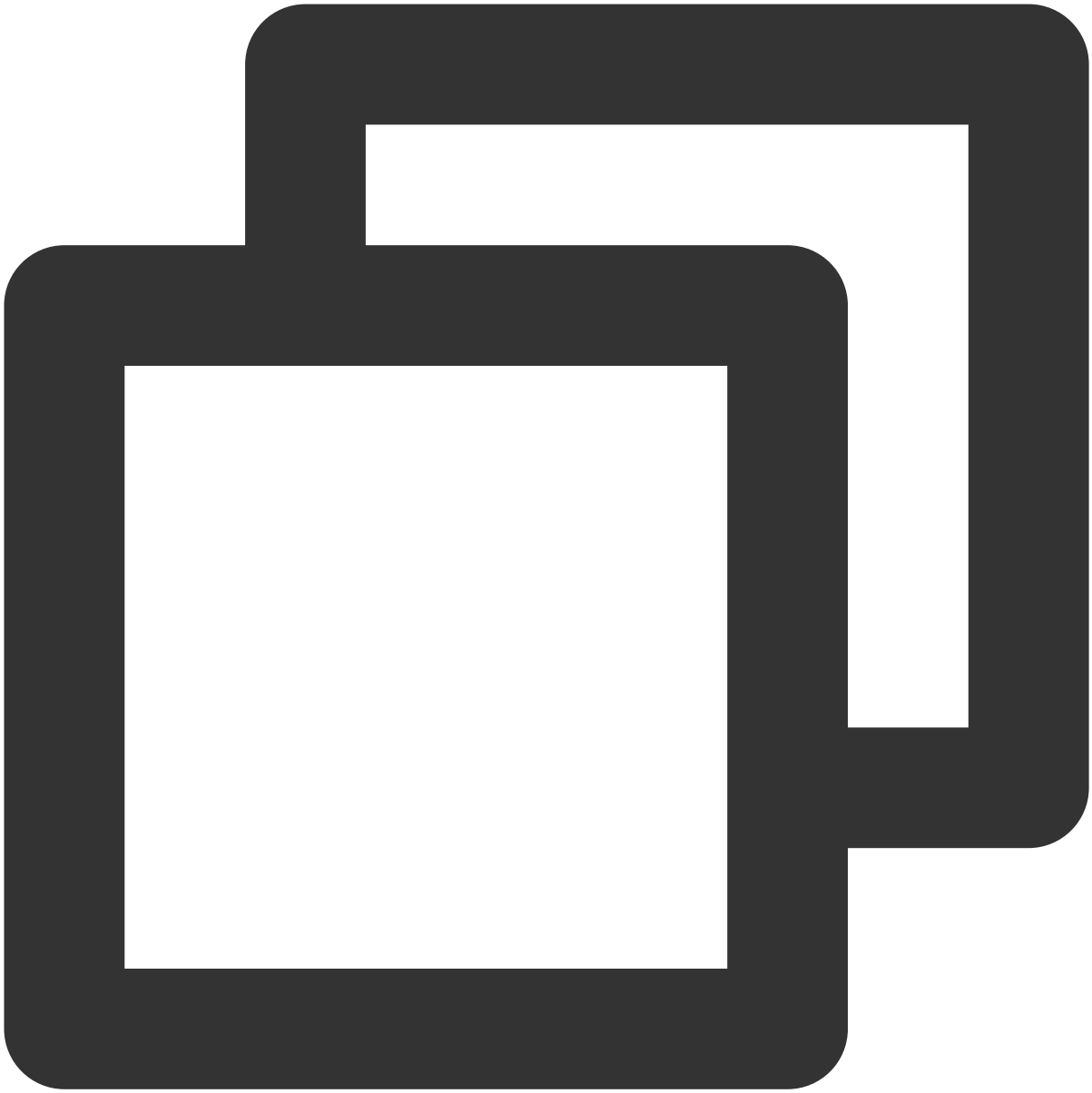
```
pod install
```

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.

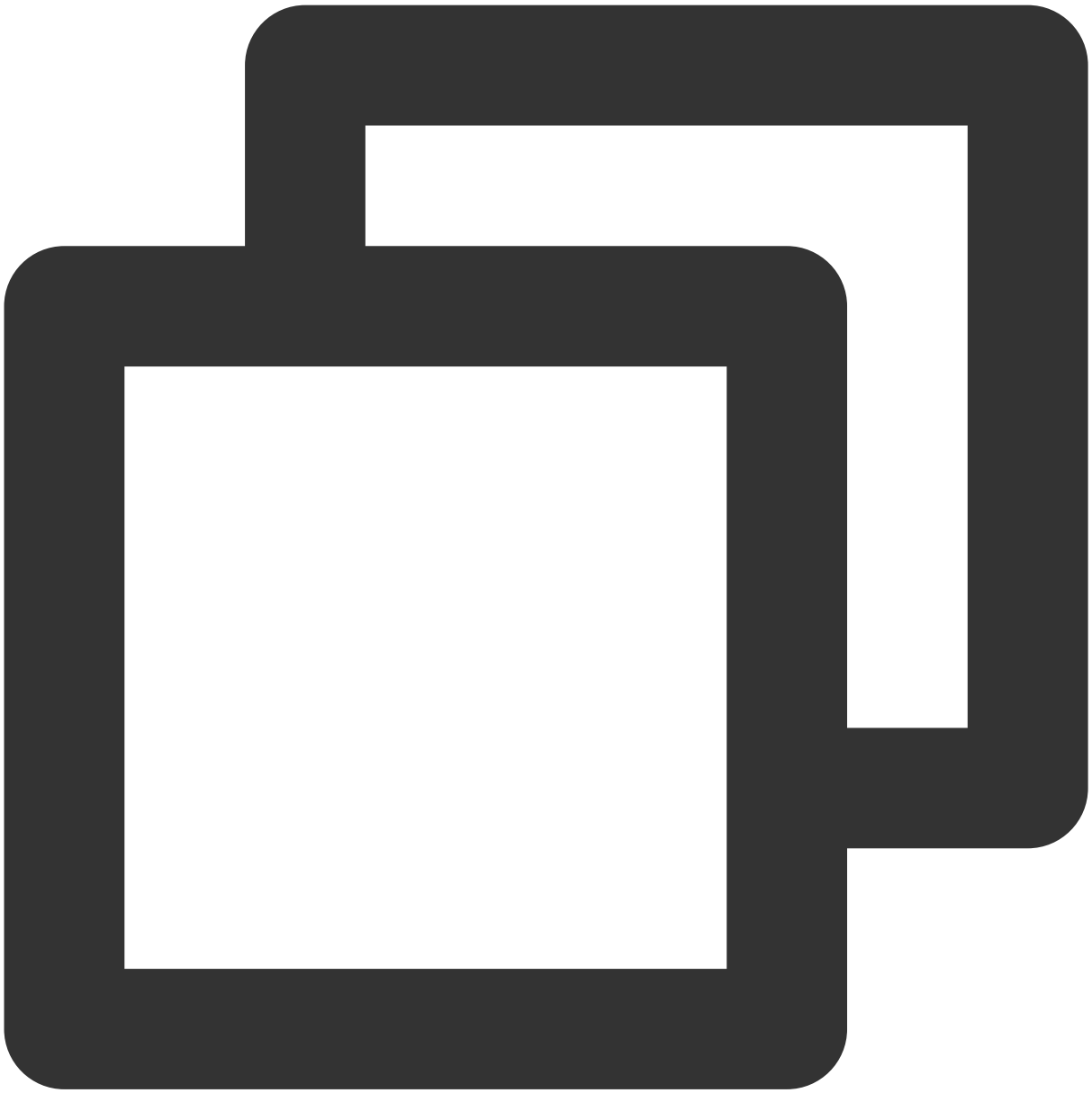


```
pod update
```

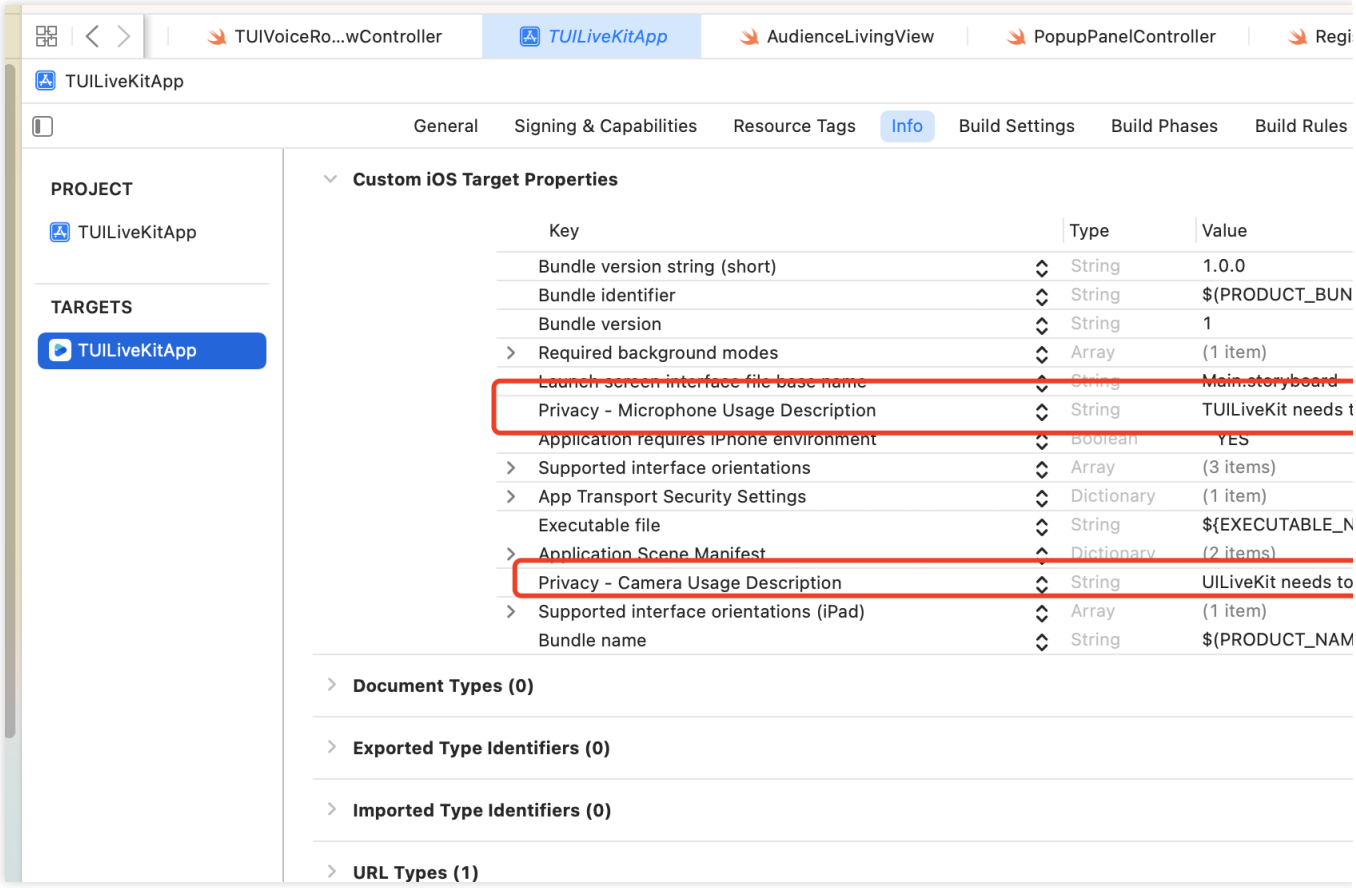
3. You can compile and run it first. If you run into problems, see Q&A. If the problem still can't be solved, you can run our Example project first. Any problems you encounter in the process of access and use, welcome to give us feedback.

Step 3. Configure the project

Your app needs mic and camera permissions to implement audio/video communication. Add the two items below to **Info** of your **App**. Their content is what users see in the mic and camera access pop-up windows.



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

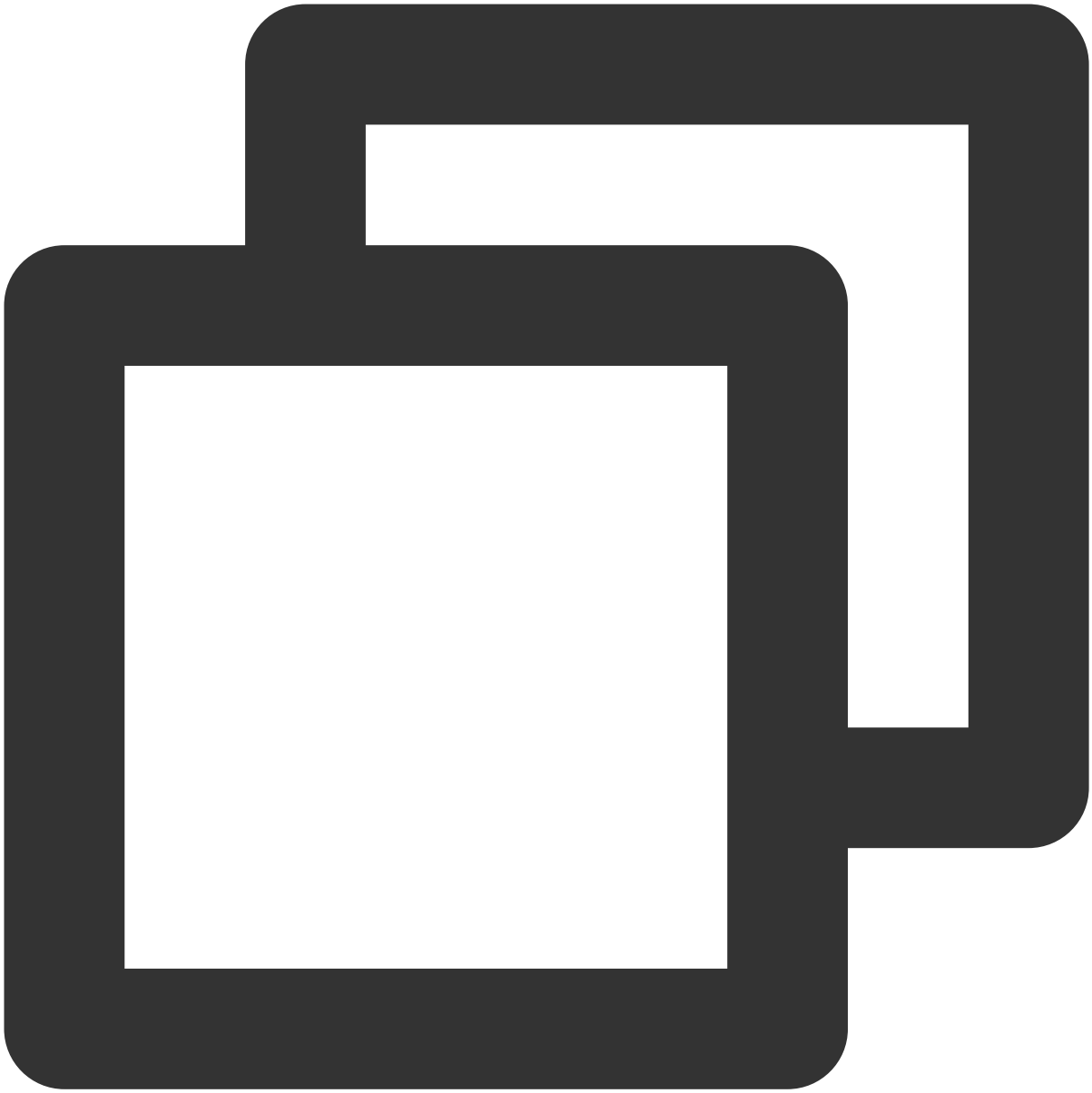


Step 4. Log in

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.

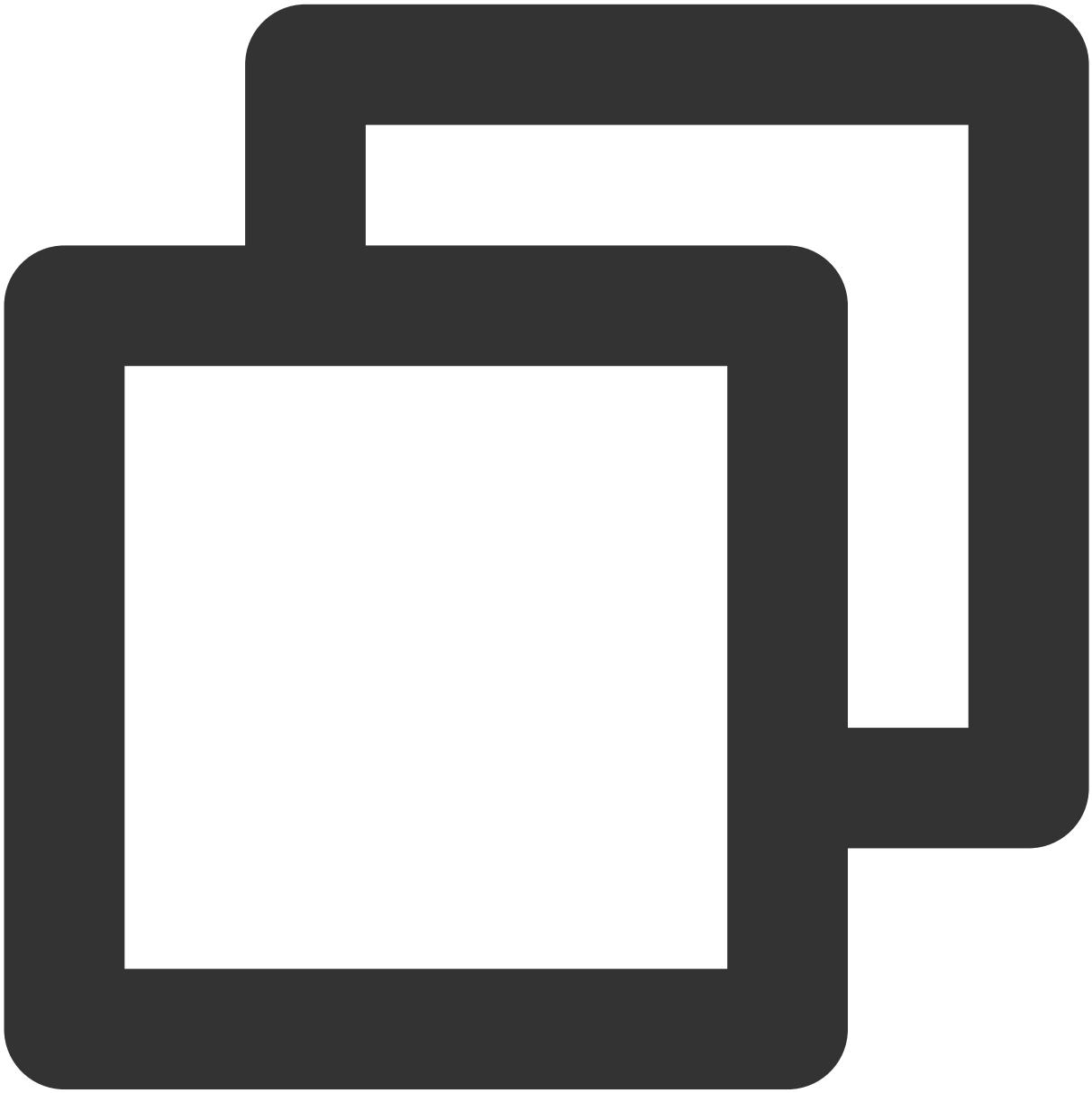
Swift

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
//
```

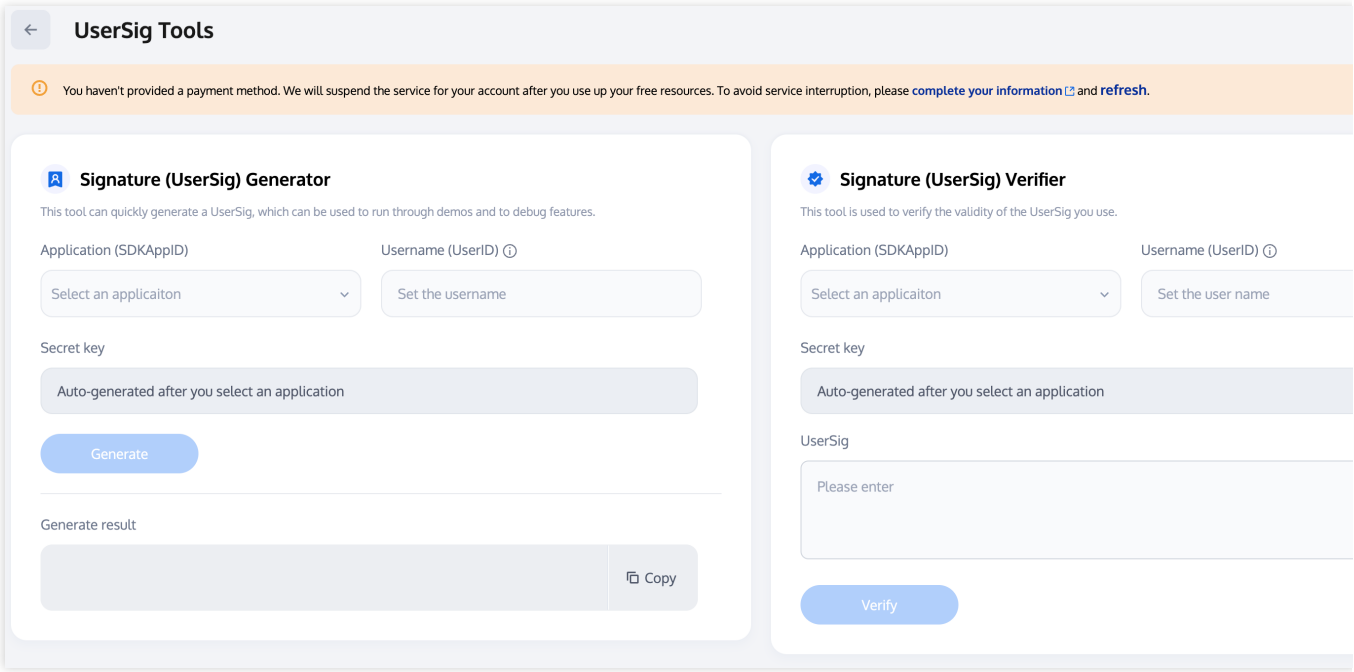
```
#import <TUICore/TUILogin.h>
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    [TUILogin login:1400000001        // Replace it with the SDKAppID obtained in S
             userID:@"denny"          // Please replace it with your UserID
            userSig:@"xxxxxxxxxxx"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

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

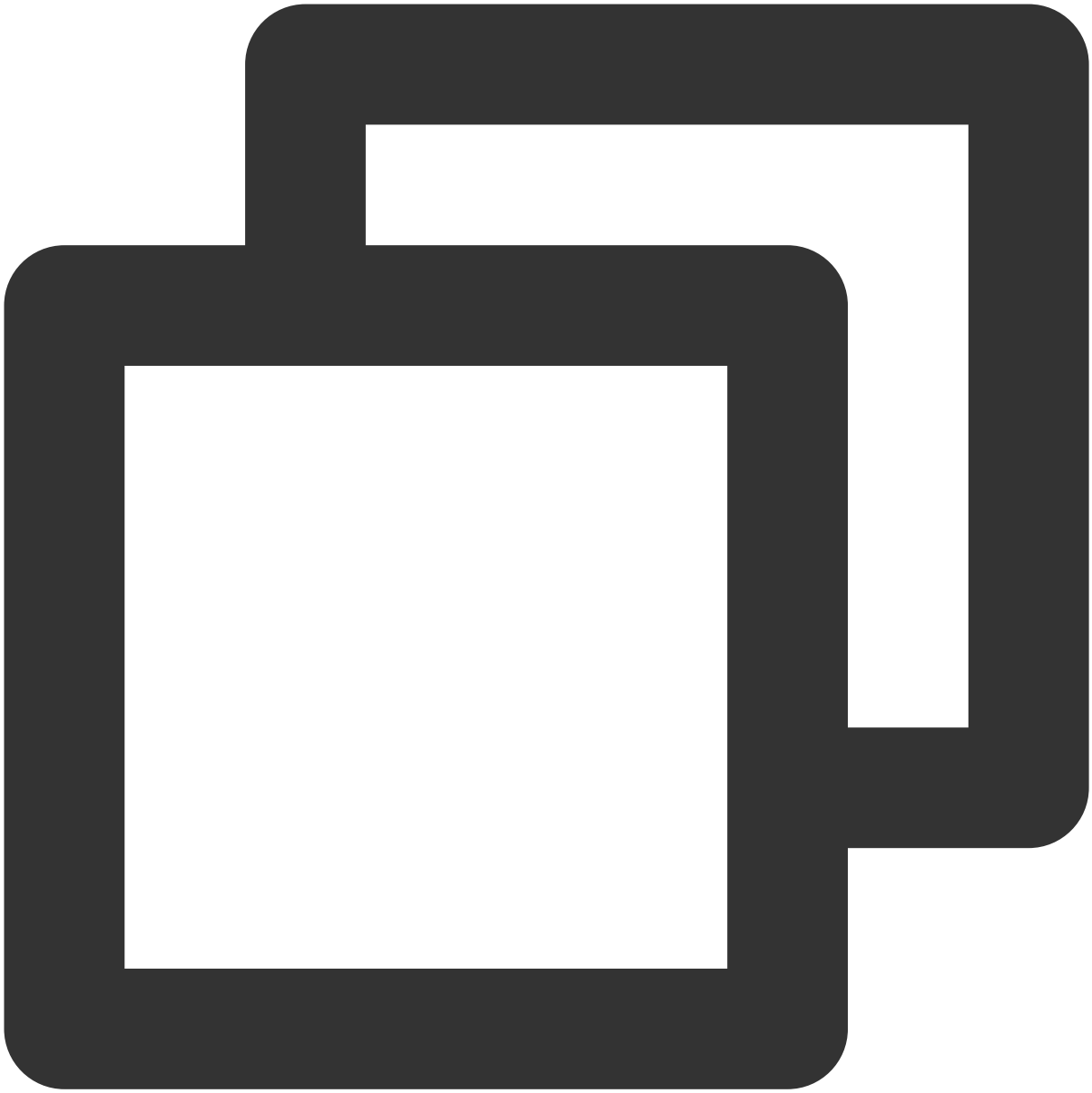
Note：

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.

By loading TUILiveKit TUILiveRoomAnchorViewController page, you can pull up preview screen, click on "start live" can be launched online video broadcast.

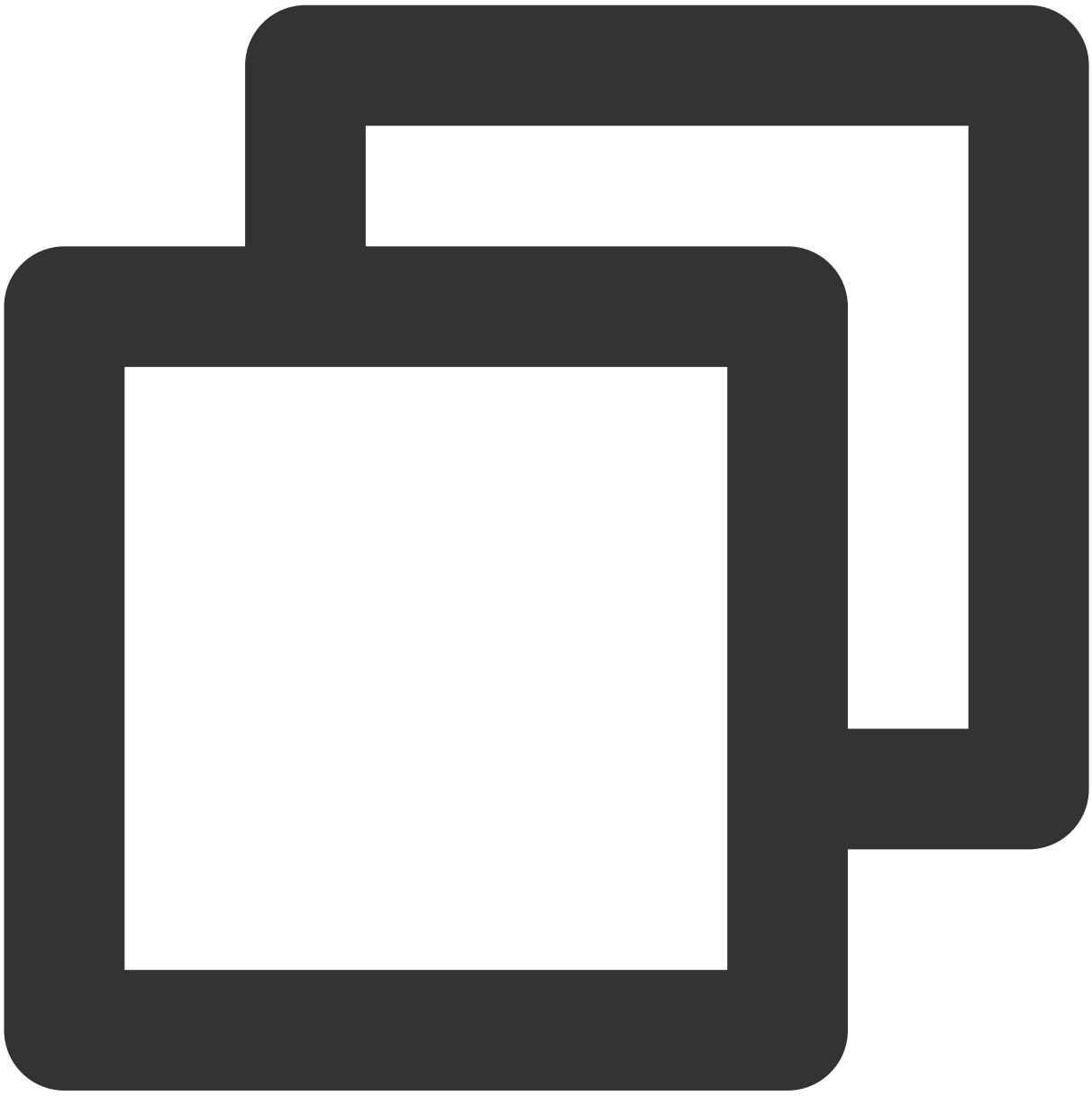
Swift

Objective-C



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

import UIKit
import TUILiveKit

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

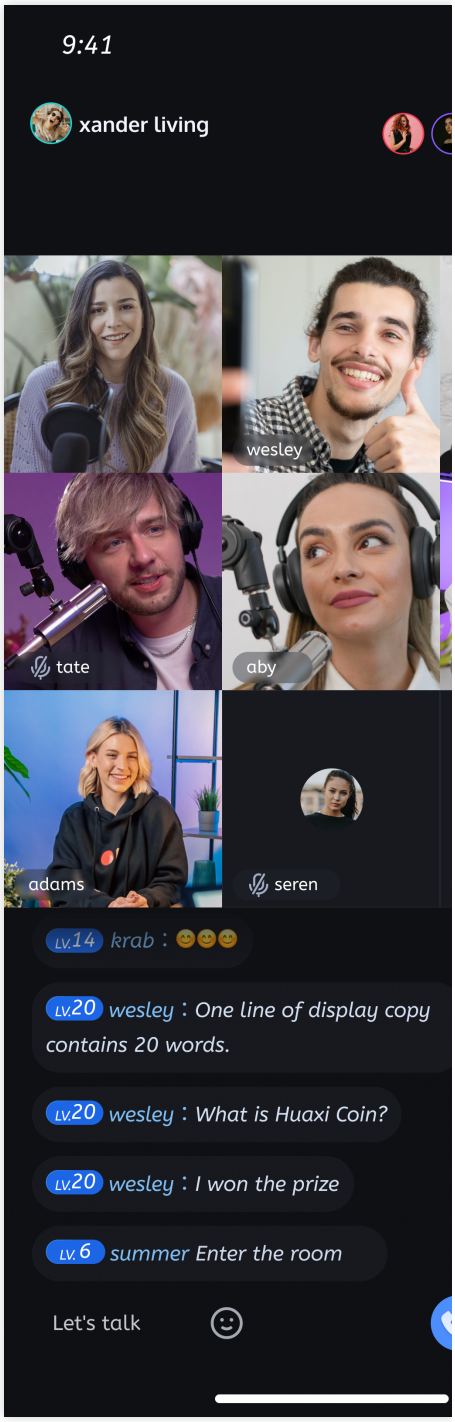
```
    // Enter the live preview screen
    let viewController = TUILiveRoomAnchorViewController(roomId: roomId)
    self.navigationController?.pushViewController(viewController, animated: true)
}
```



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

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

```
- (void)buttonTapped:(UIButton *)sender {
    //RoomId can be customized, Recommended use [LiveIdentityGenerator.shared genera
    NSString *roomId = @"123666";

    // Enter the live preview screen
    TUILiveRoomAnchorViewController *liveViewController = [[TUILiveRoomAnchorViewCo
    [self.navigationController pushViewController:liveViewController animated:true]
}
```

Video Live Preview Screen



Live video streaming with pictures

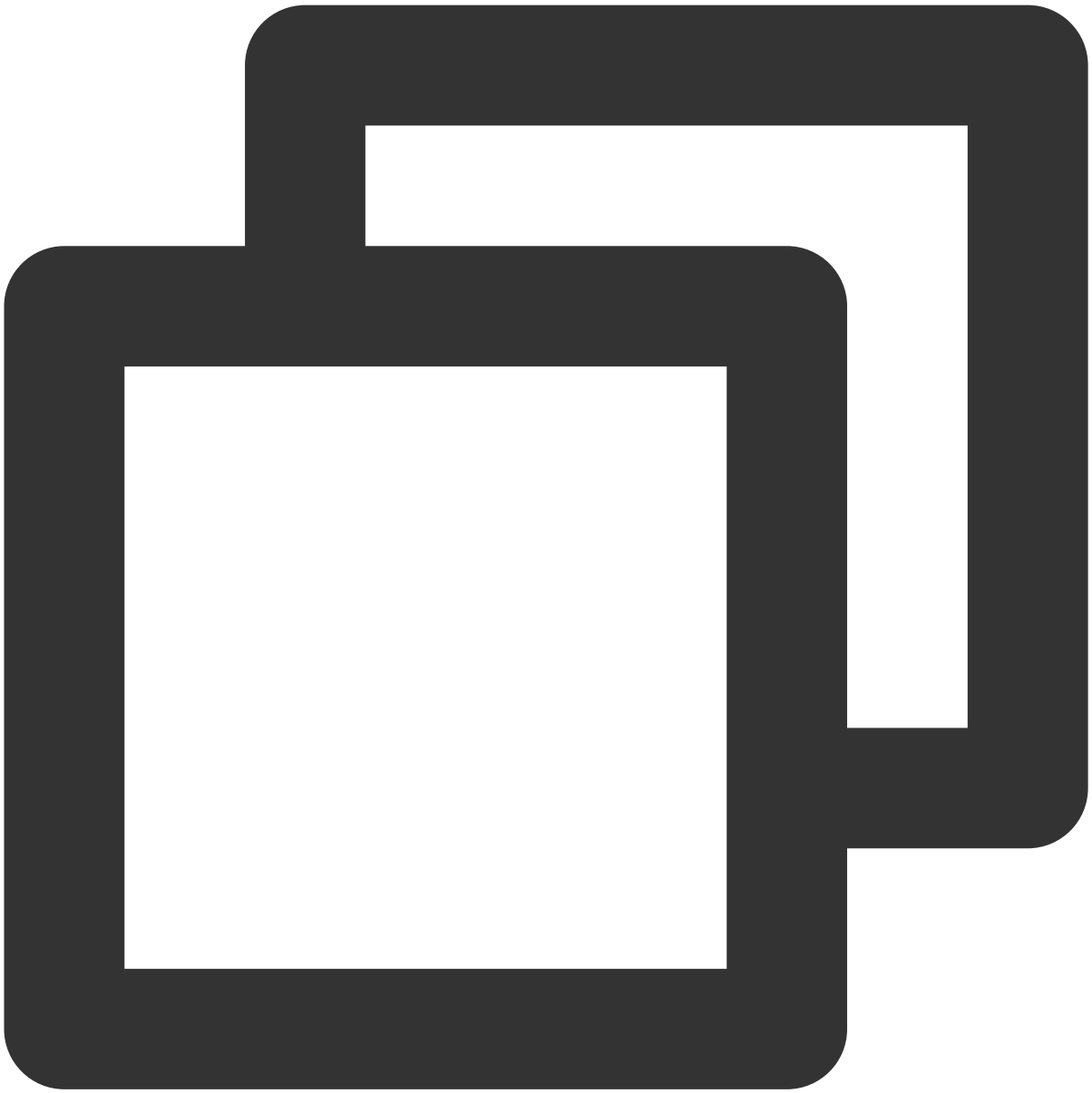
Step 6. Pull the room list

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.

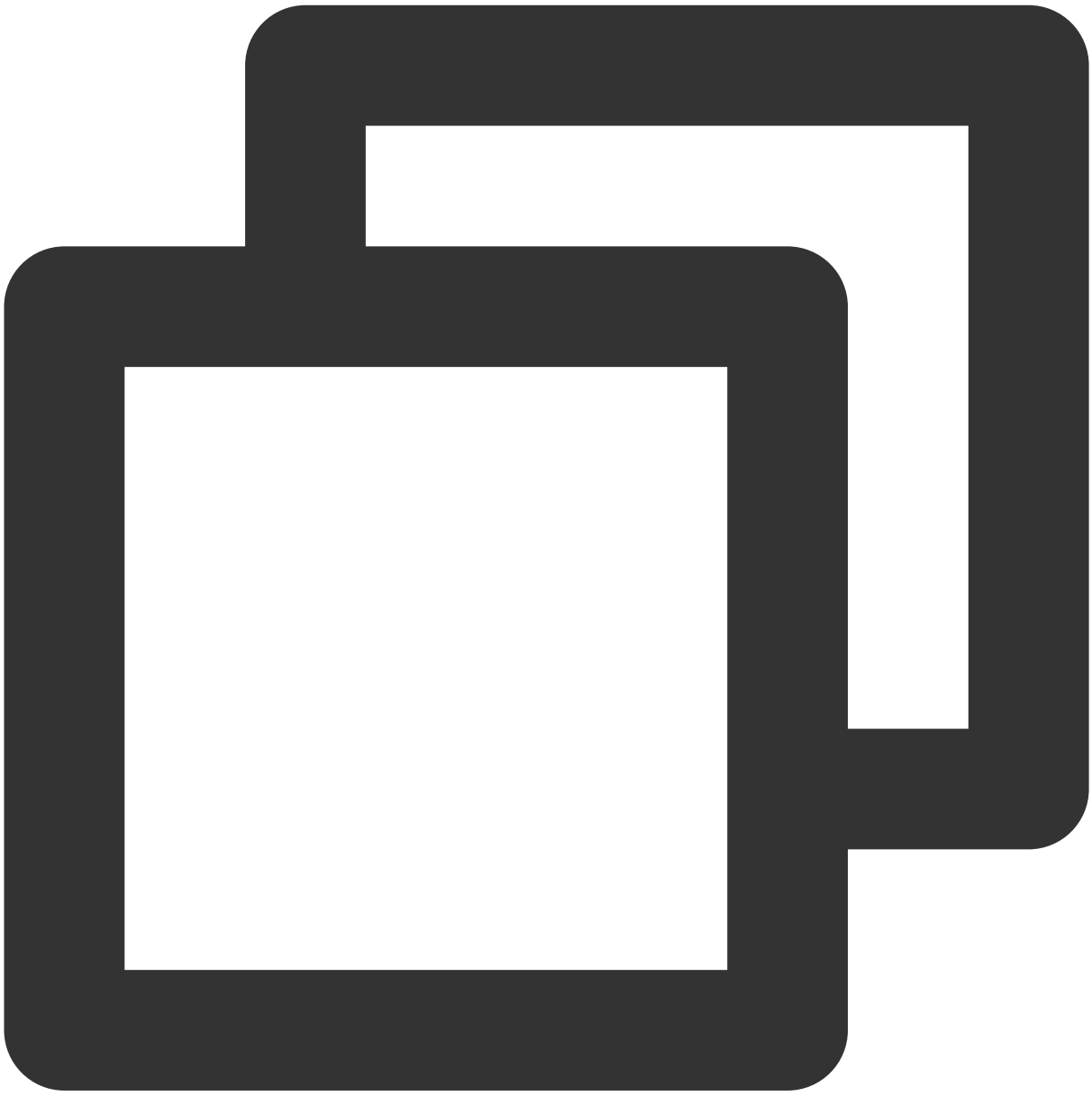
Swift

Objective-C



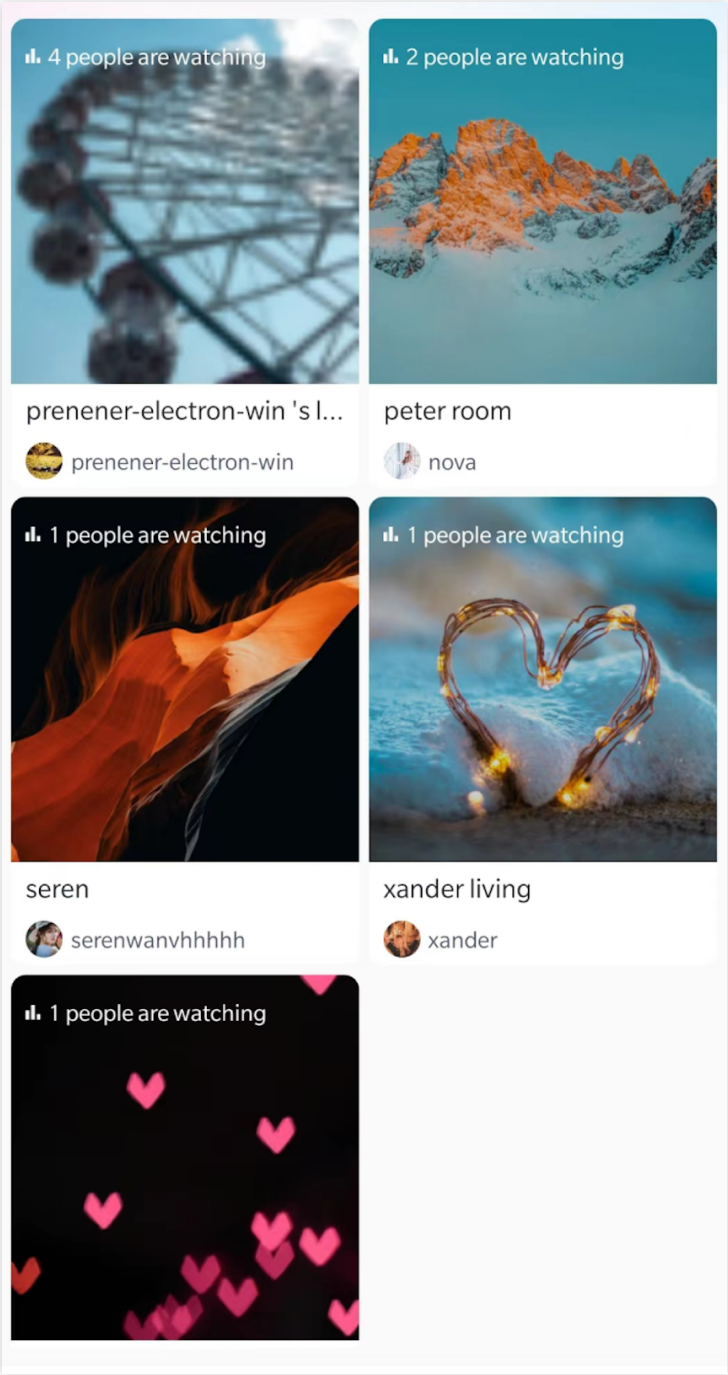
```
//
//  MainViewController.swift
//

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>
```

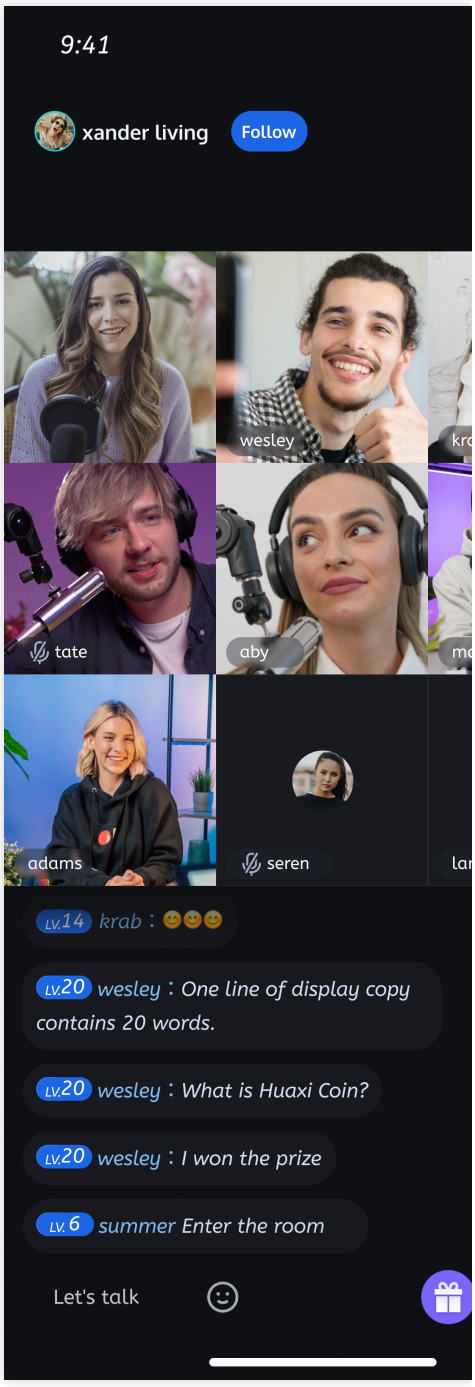
```
- (void)buttonTapped:(UIButton *)sender {
    // Enter room list
    TUILiveListViewController *liveListViewController = [[TUILiveListViewController
    [self.navigationController pushViewController:liveListViewController animated:t
}
```



Step 7. The audience enters the studio

On the room list page of Step 6, click any room to automatically enter the live broadcast room.

Video Live Room



Video Live Room

More features

Interactive Bullet Comments

Interactive Gifts
Gift Effects
Beauty Effects

Q&A

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

Suggestions and Feedback

If you have any suggestions or feedback, please contact info_rtc@tencent.com.

Android

Last updated：2024-08-02 16:58:15

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

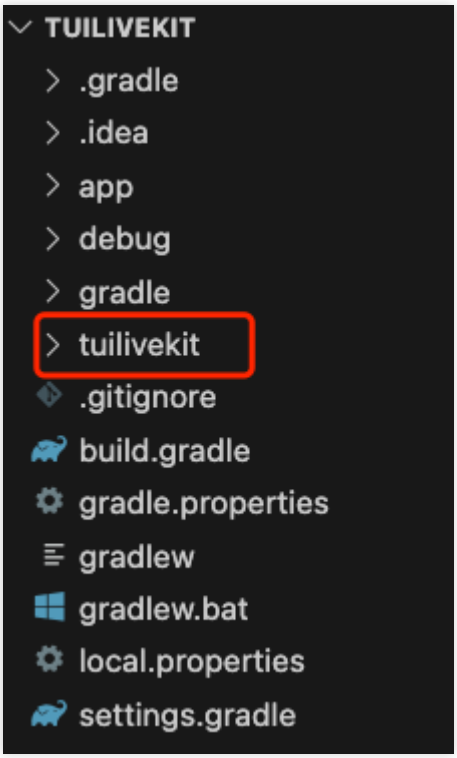
Android 5.0 (SDK API level 21) or above.
Android Studio 4.2.1 or above.
Devices with Android 5.0 or above.
If you have any questions during environment configuration or compilation and running, please refer to the FAQ.

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

Java
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

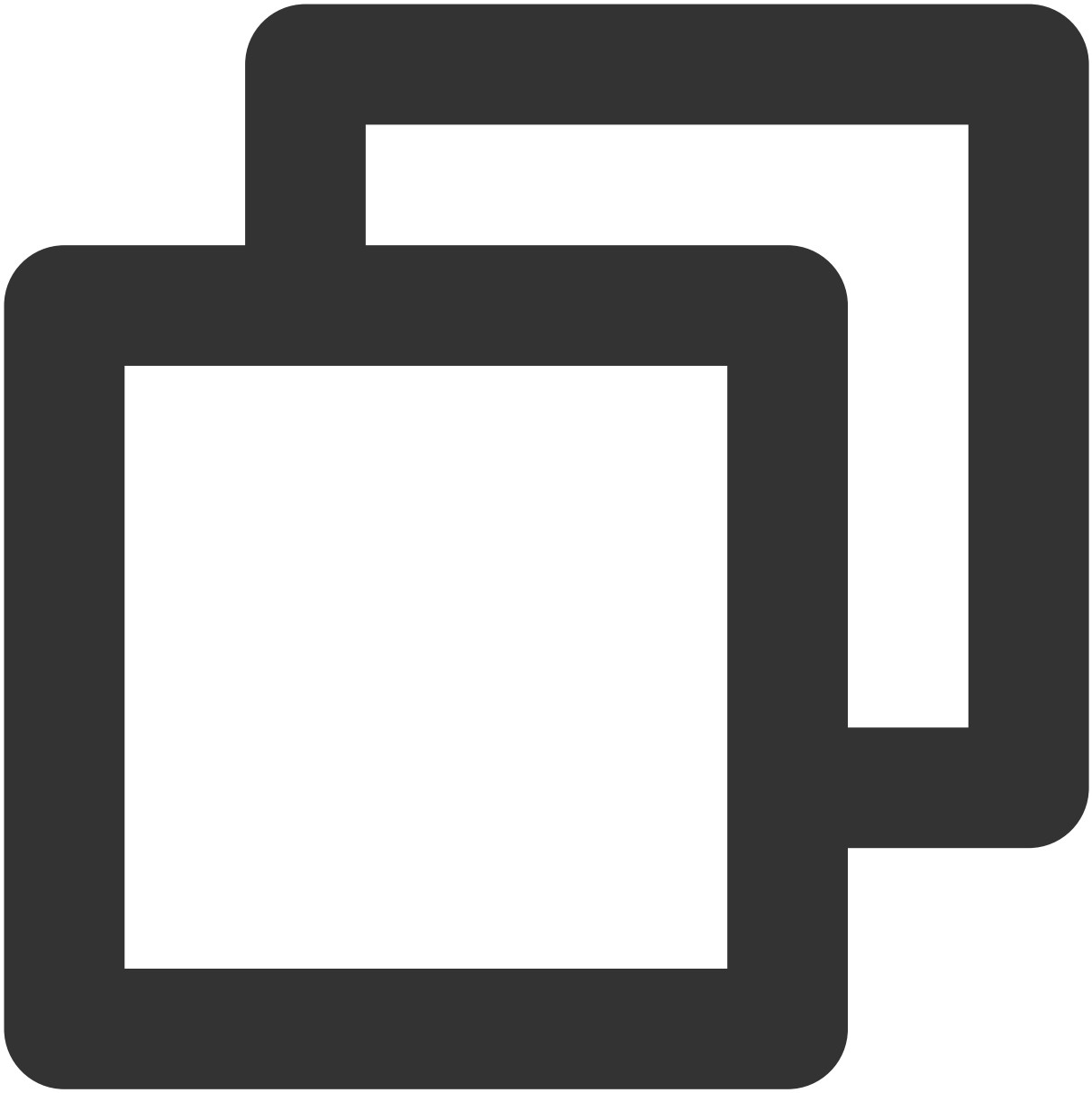
Java

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

Gradle 7.0 earlier

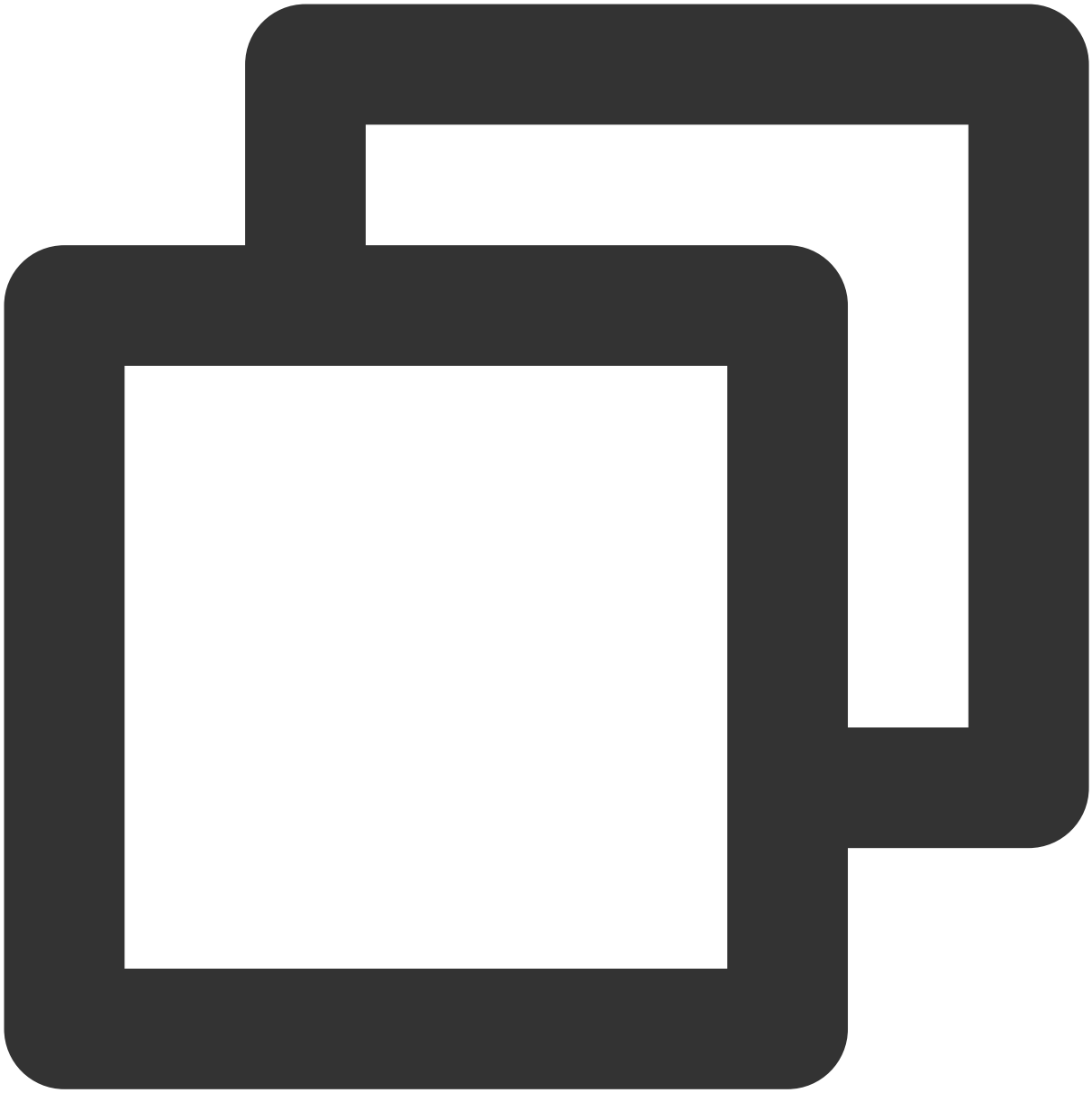
gradle 7.0 or later

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



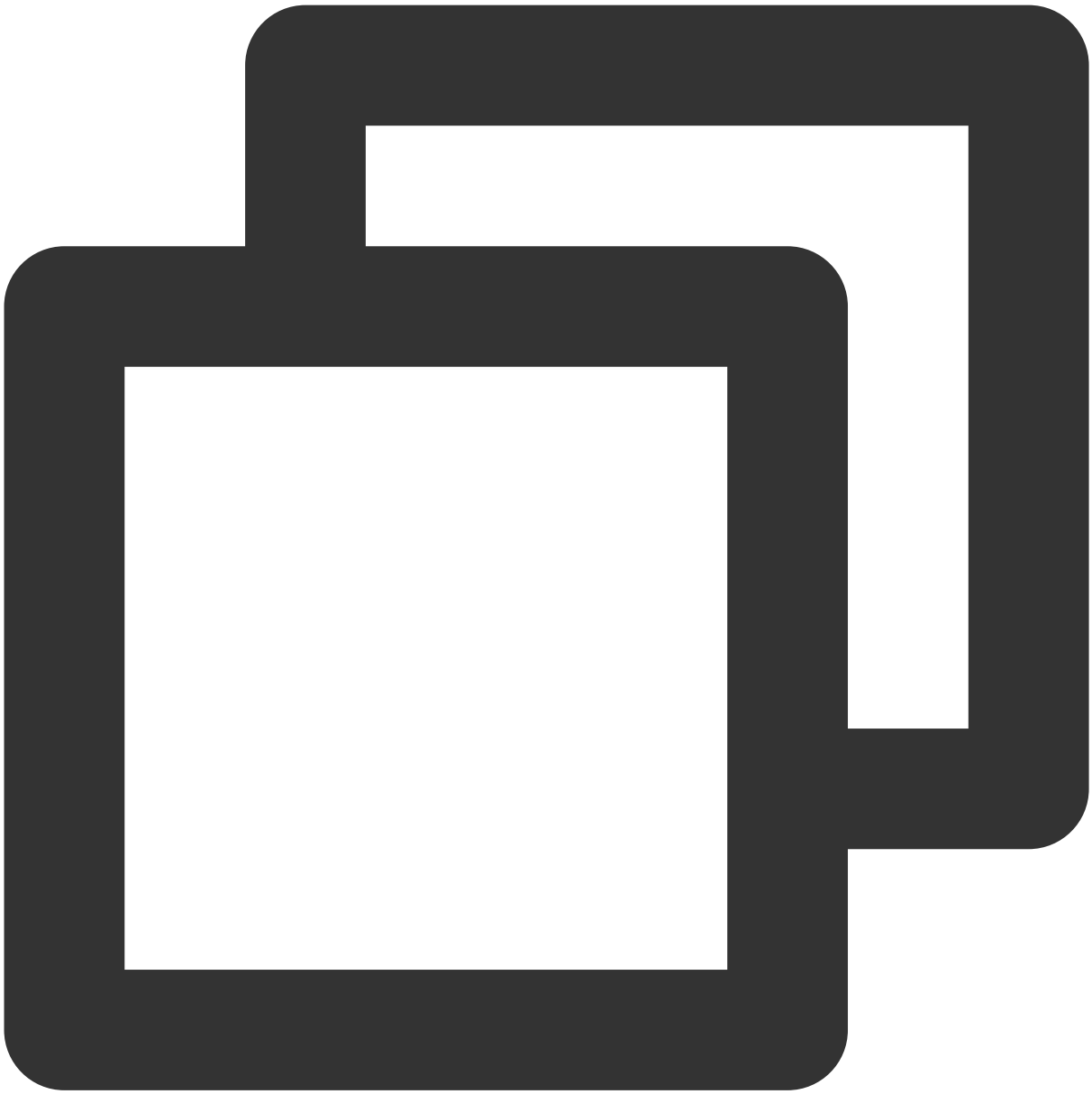
```
allprojects {
    repositories {
        google()
        mavenCentral()
        // add jitpack repository
        maven { url 'https://jitpack.io' }
    }
}
```

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



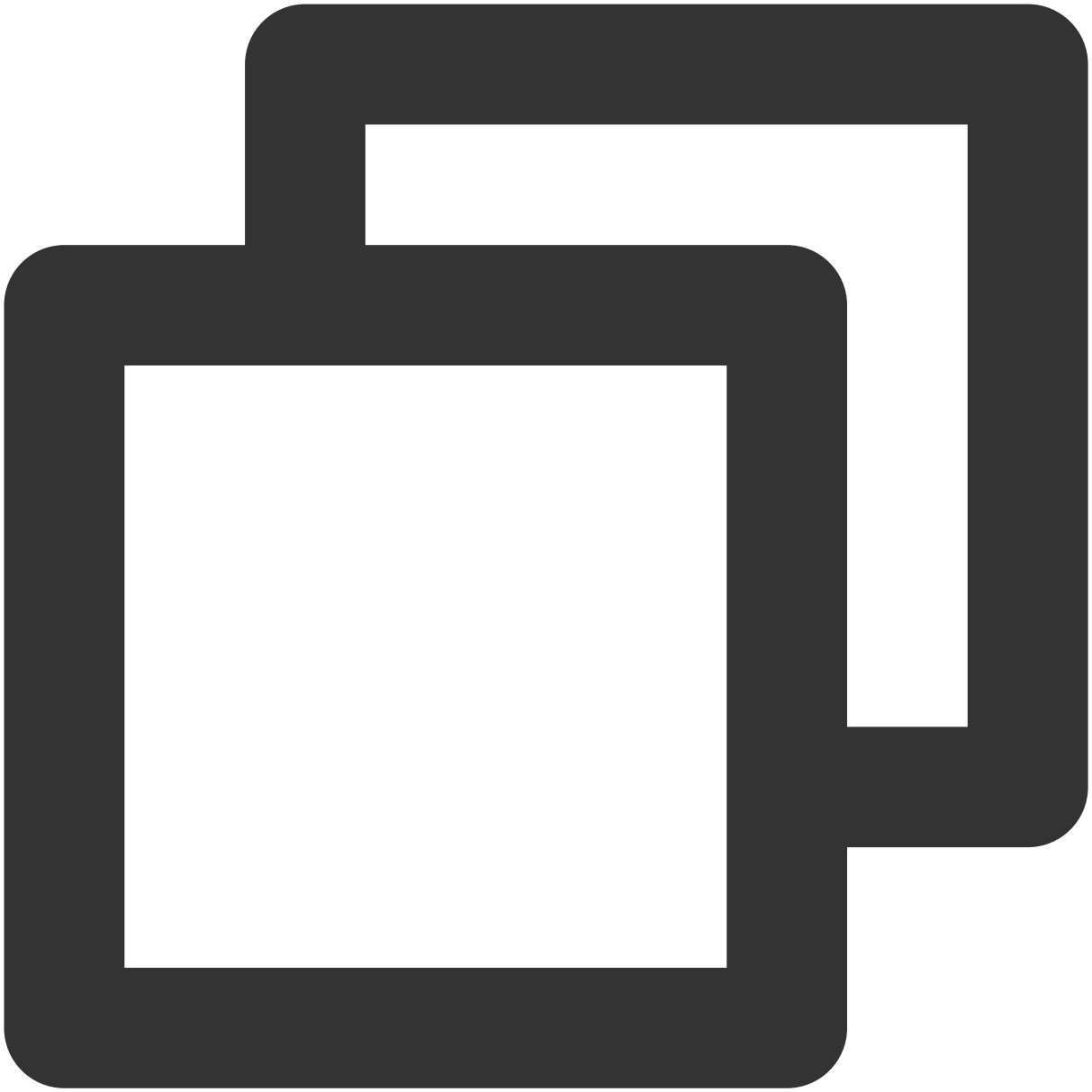
```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        // add jitpack repository
        maven { url 'https://jitpack.io' }
    }
}
```

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:

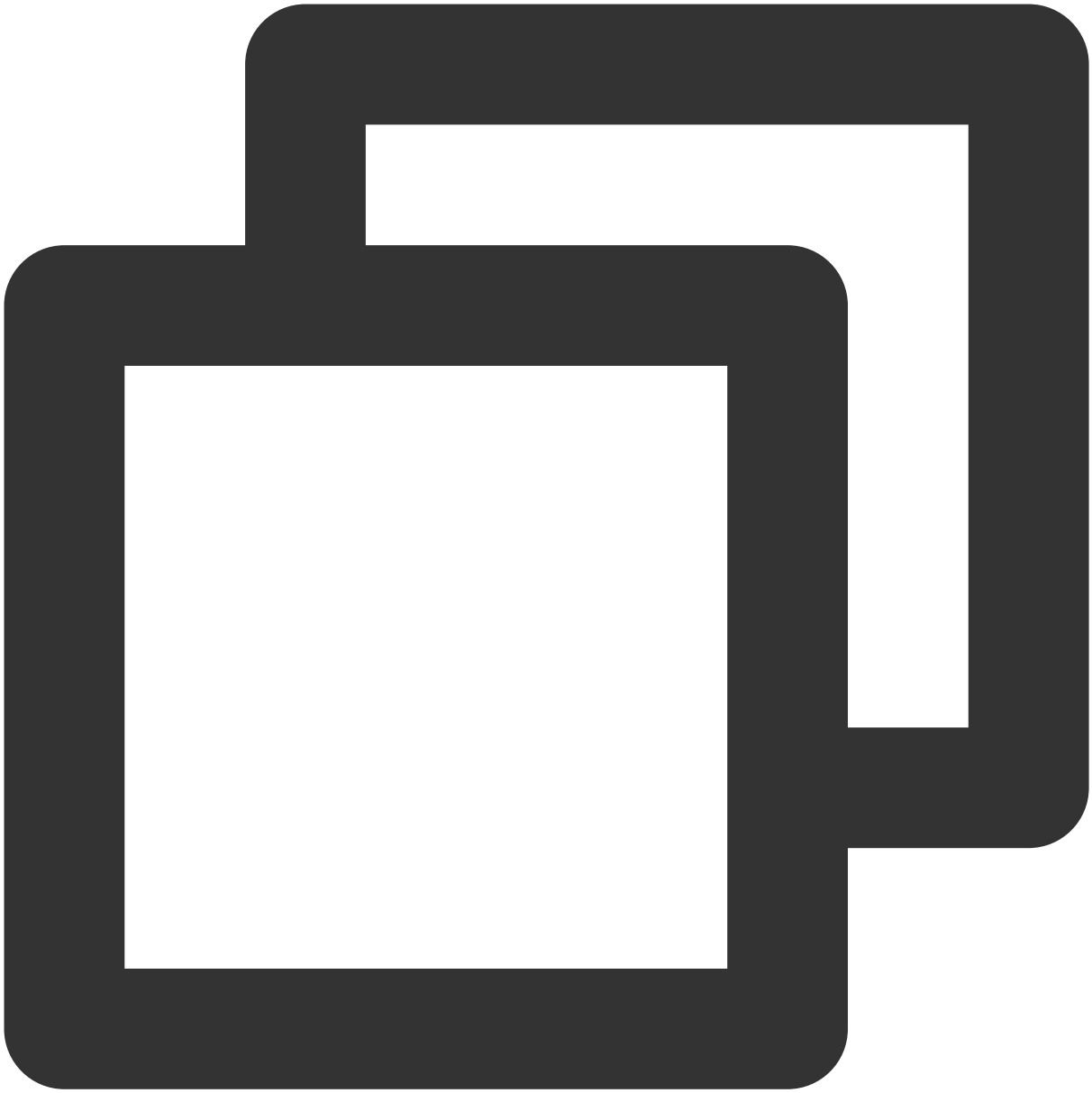


```
api project(':tuilivekit')
```

Note:

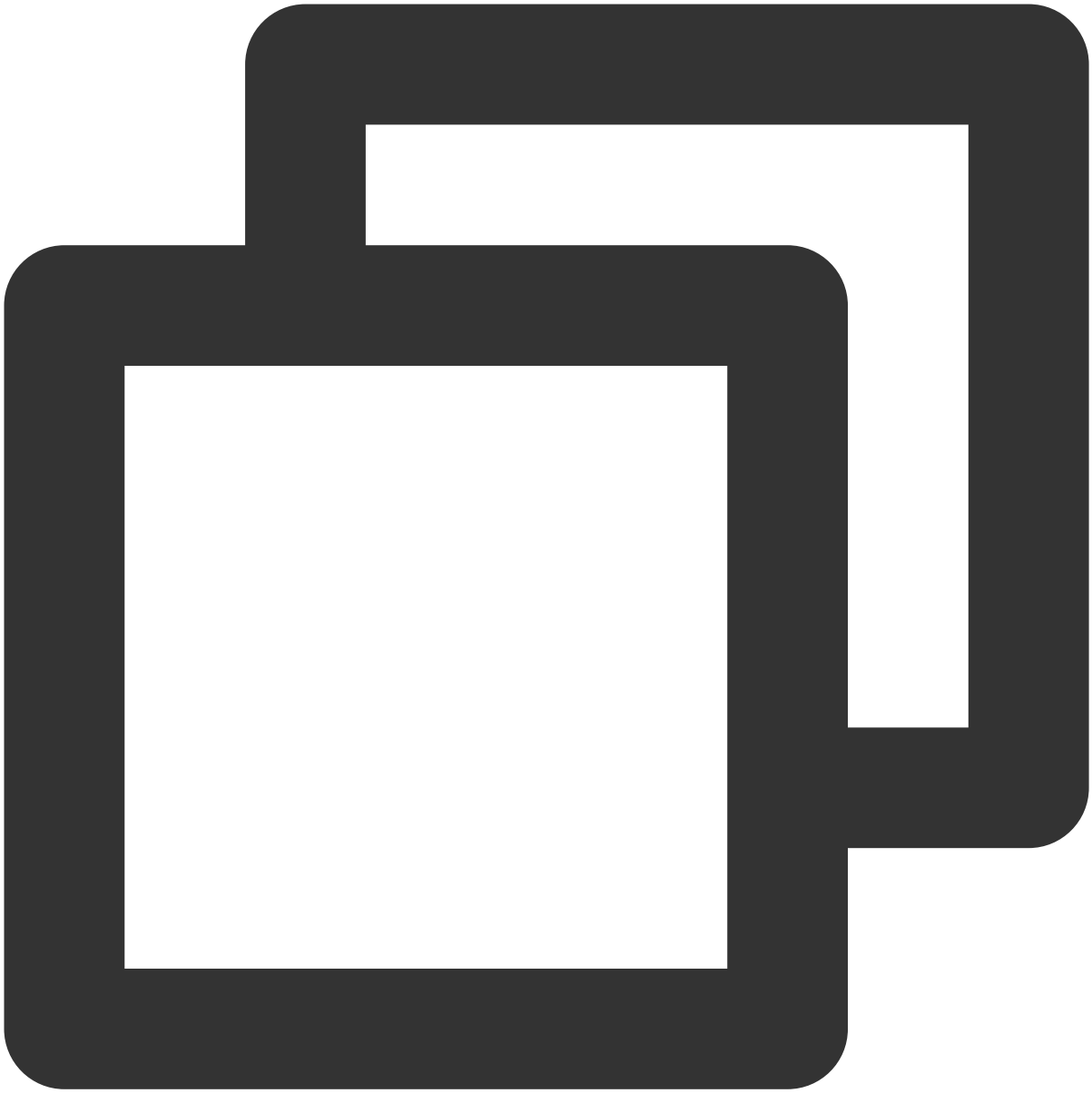
The TUILiveKit project has internal dependencies by default: `TRTC SDK`、`IM SDK`、`tuiroomengine` and the public library `tuicore`, and does not require separate configuration by developers. If you need to upgrade the version, just modify the `tuilivekit/build.gradle` file.

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` :



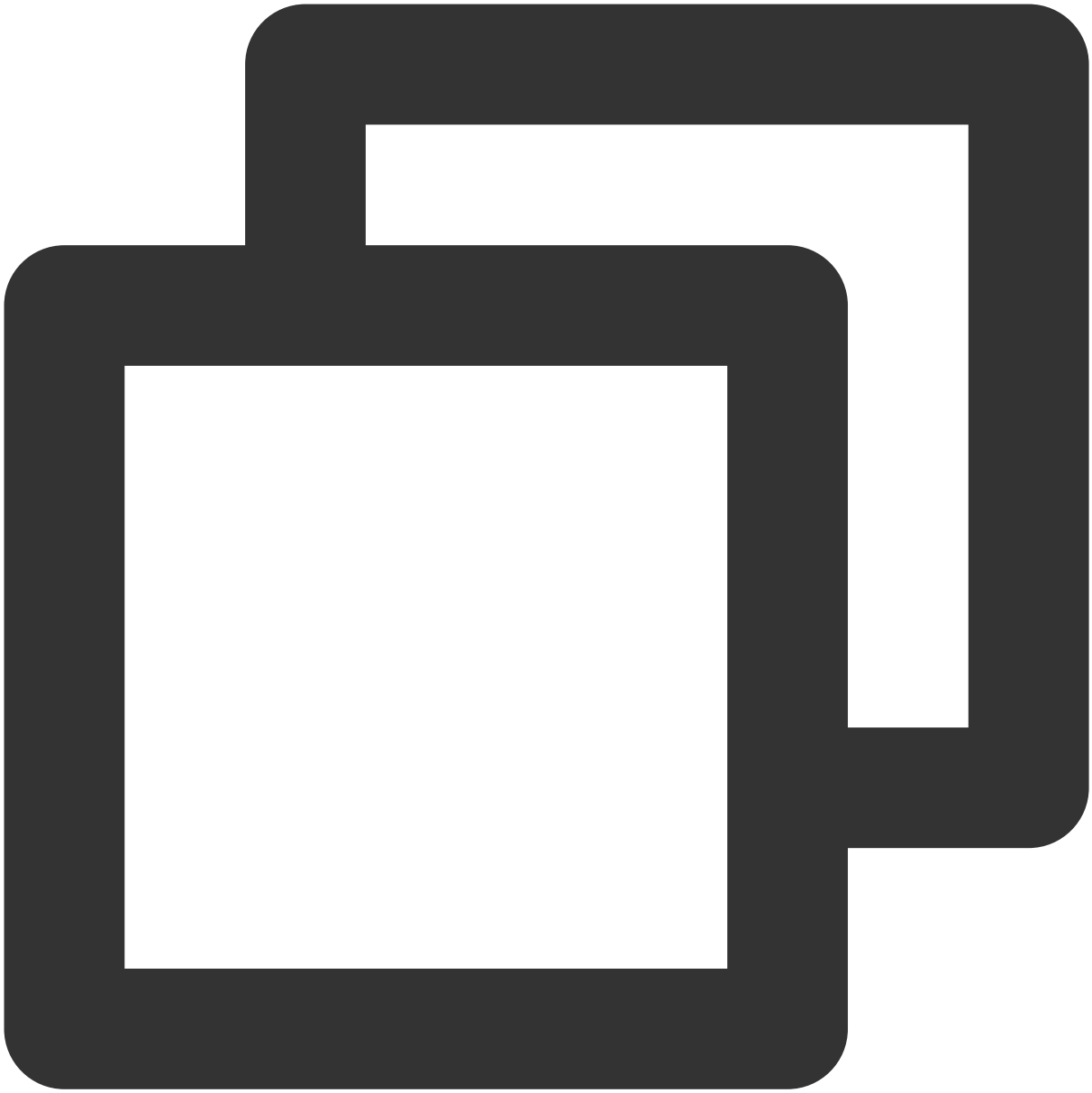
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
        ...
    </application>
</manifest>
```

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 Questions.

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



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

    @Override
```

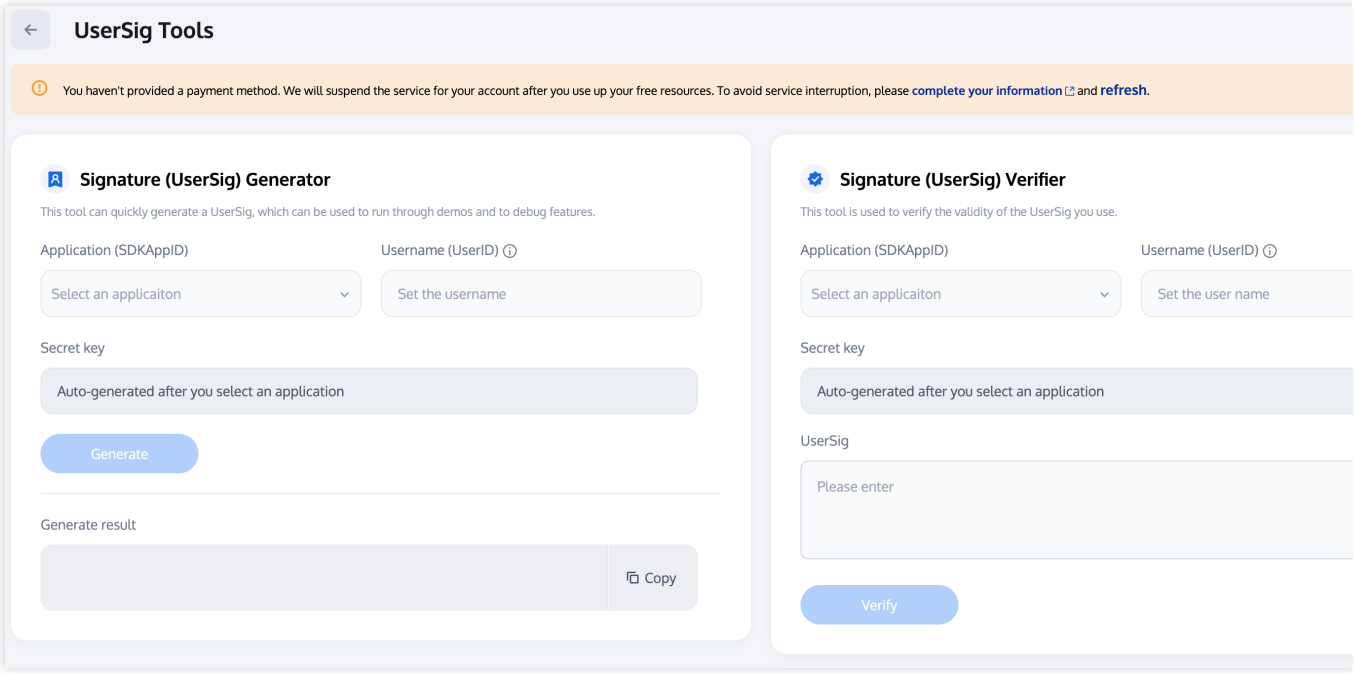
```
    public void onError(int errorCode, String errorMessage) {
        Log.e(TAG, "login failed, errorCode: " + errorCode + " msg:" + errorMessage
    }
});
```

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.

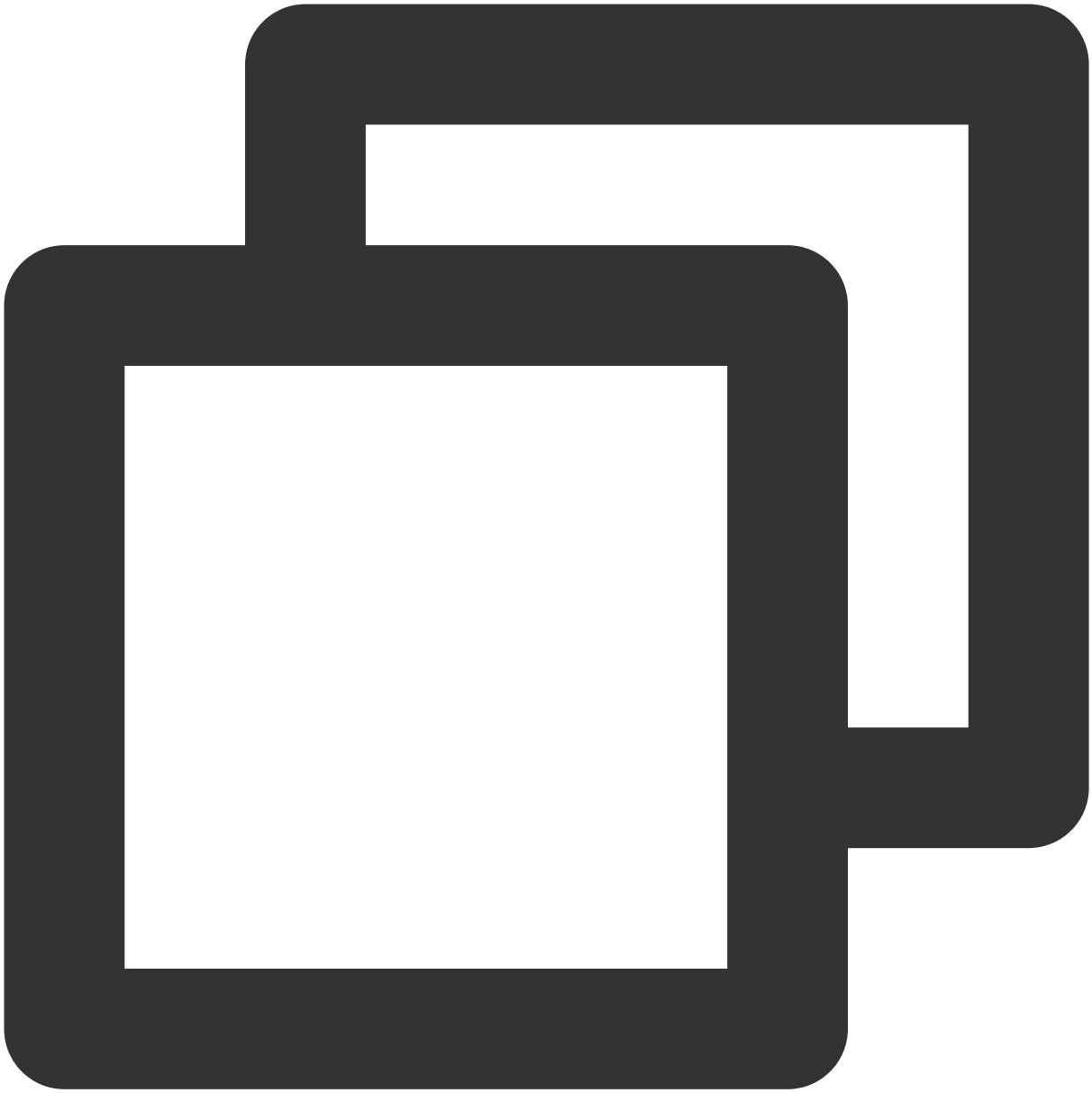
The sample code on GitHub uses the `genTestUserSig` function to calculate `UserSig` locally, so as to help you complete the current integration process more quickly. However, this scheme exposes your `SecretKey` in the application code, which makes it difficult for you to upgrade and protect your `SecretKey` 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.

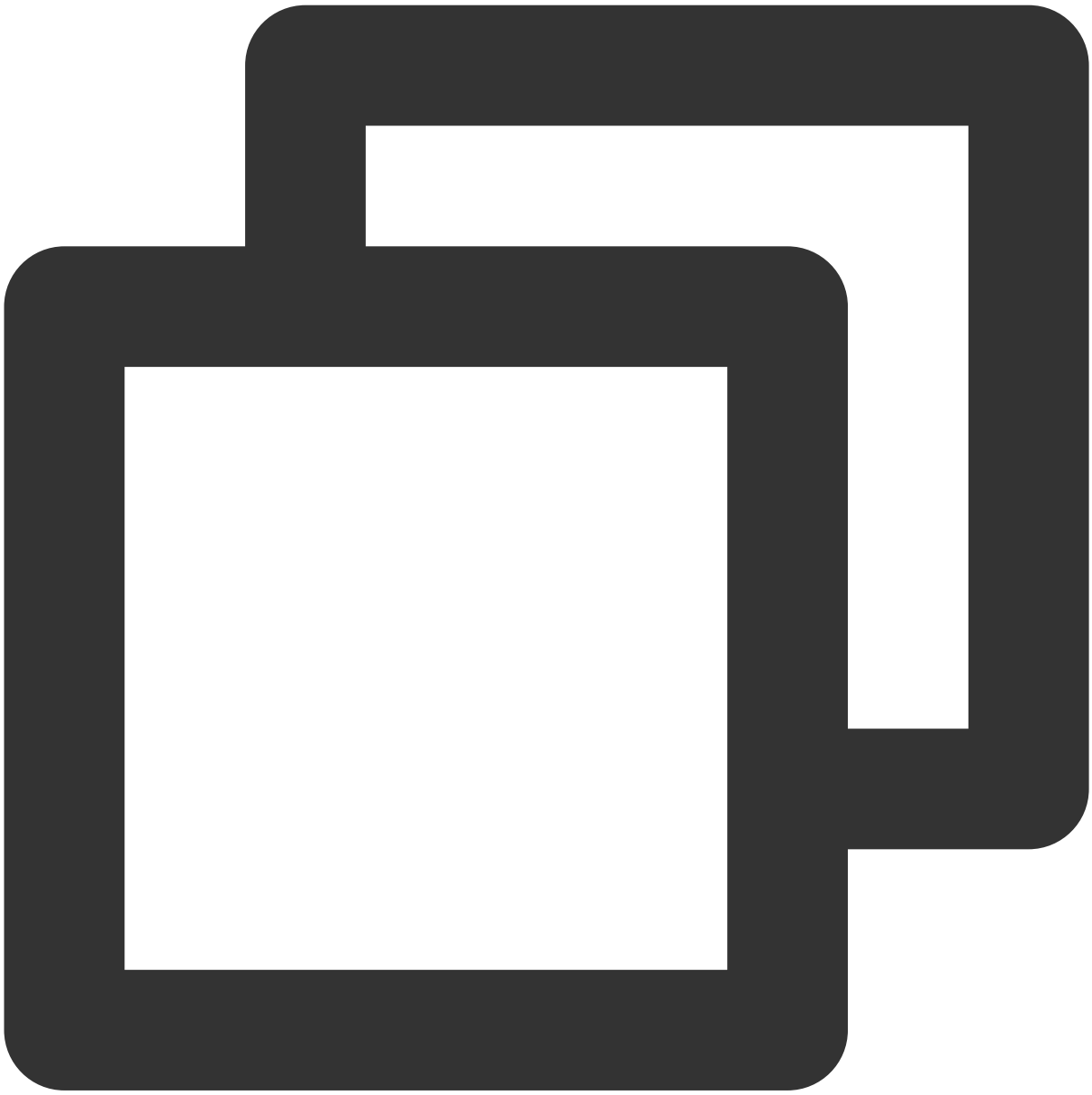
1. Create a new file named app_activity_start_live.xml (Default path: app/src/main/res/layout/app_activity_start_live.xml).



```
<?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 `StartLiveActivity.java` and register in the `AndroidManifest.xml` . By loading TUILiveKit `TUILiveAnchorFragment` page, you can pull up preview screen.

java



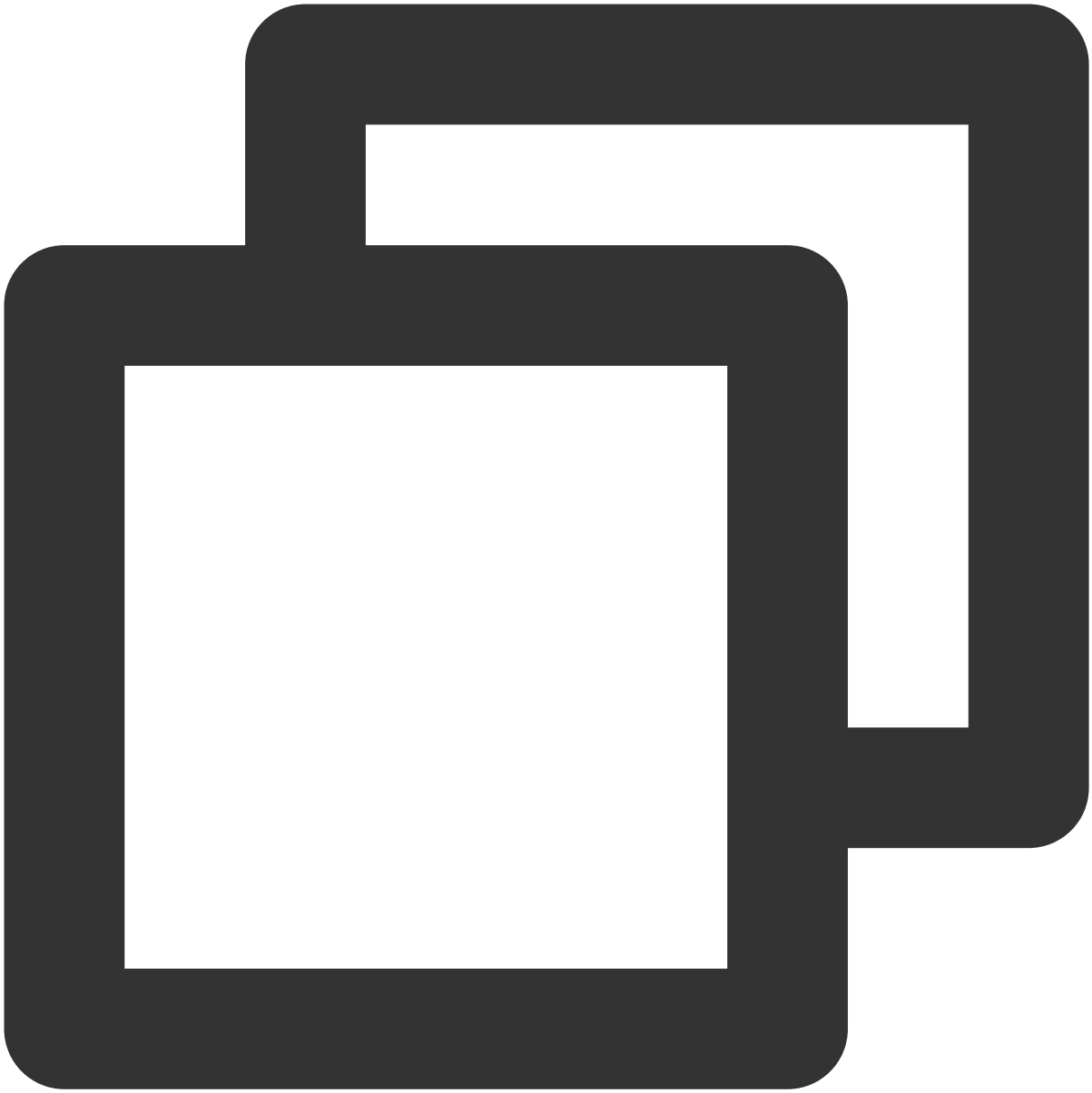
```
public class StartLiveActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.app_activity_anchor);

        FragmentManager fragmentManager = getSupportFragmentManager();
```

```
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction(
        LiveIdentityGenerator identityGenerator = LiveIdentityGenerator.getInstance
        String liveRoomId = identityGenerator.generateId(TUILogin.getUserId(), Room
        String voiceRoomId = identityGenerator.generateId(TUILogin.getUserId(), Roo
        TUILiveAnchorFragment anchorFragment = new TUILiveAnchorFragment(liveRoomId
        fragmentTransaction.add(R.id.fl_container, anchorFragment);
        fragmentTransaction.commit();
    }
}
```

Note:

`TUILiveAnchorFragment` needs to pass in two parameters: `liveRoomId` and `voiceRoomId`, the naming convention of liveRoomId is "live_xxx", and the naming convention of voiceRoomId is "voice_xxx". You can use the `generateId` method of the utility class `LiveIdentityGenerator` to help generate the corresponding RoomId. For example, if you pass "123456" and RoomType.LIVE to the generateId method, you will get a RoomId of "live_123456".

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



```
<?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 StartLiveActivity, please use your actual package
        <activity
            android:name="com.trtc.uikit.livekit.example.view.main.StartLiveActivit
            android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"/>
        ...
    </application>
```

```
</manifest>
```

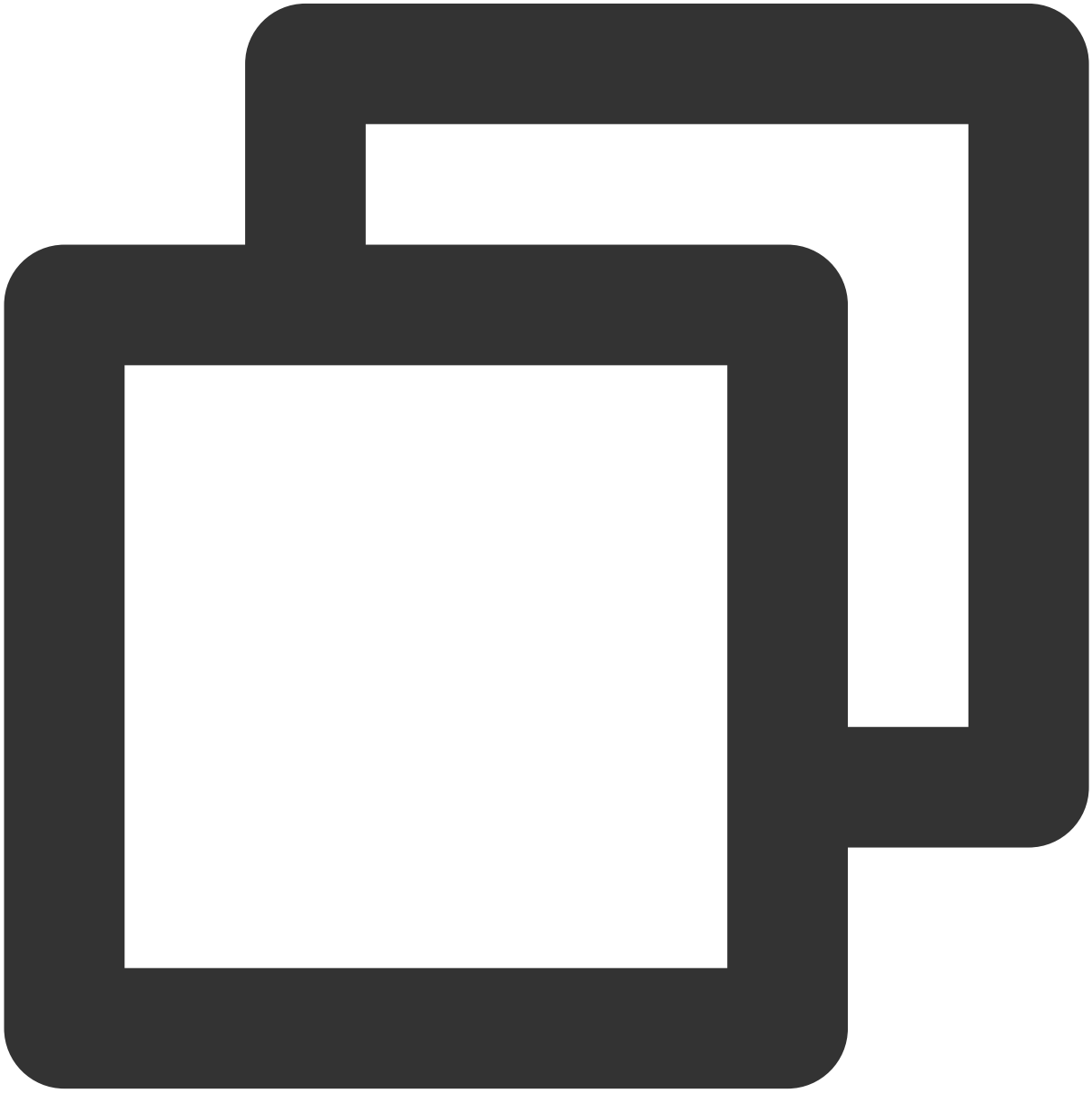
Note:

Since `StartLiveActivity` inherited from `AppCompatActivity` , `StartLiveActivity` 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 Questions.

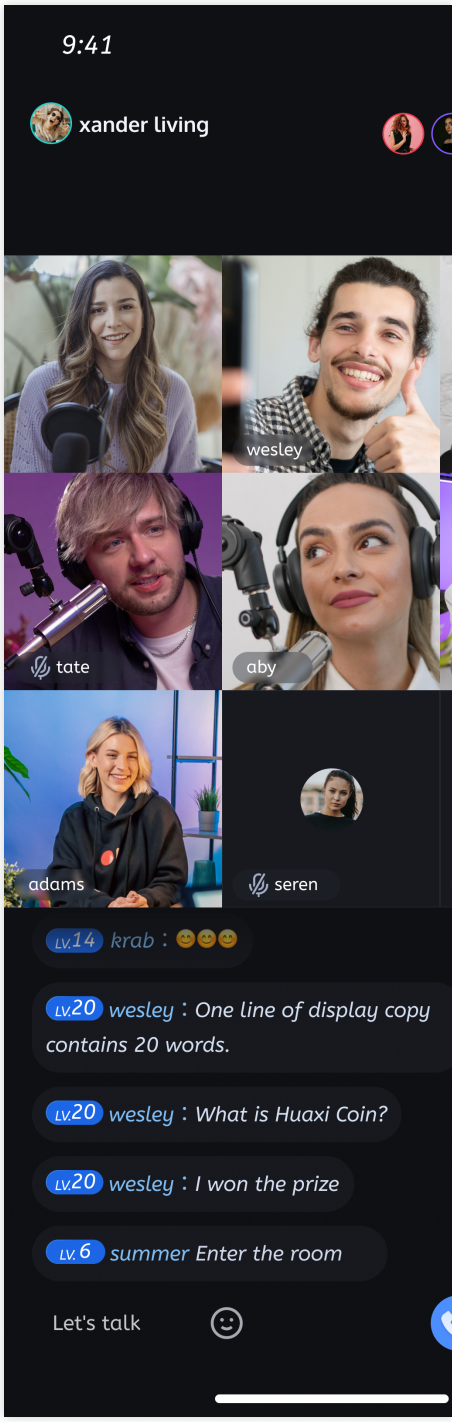
3. Where you need to start live streaming (depending on your business, it can be executed in a click event in MainActivity by default), perform the following operations to pull up the host start page:

Java



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

Video Live Preview Screen



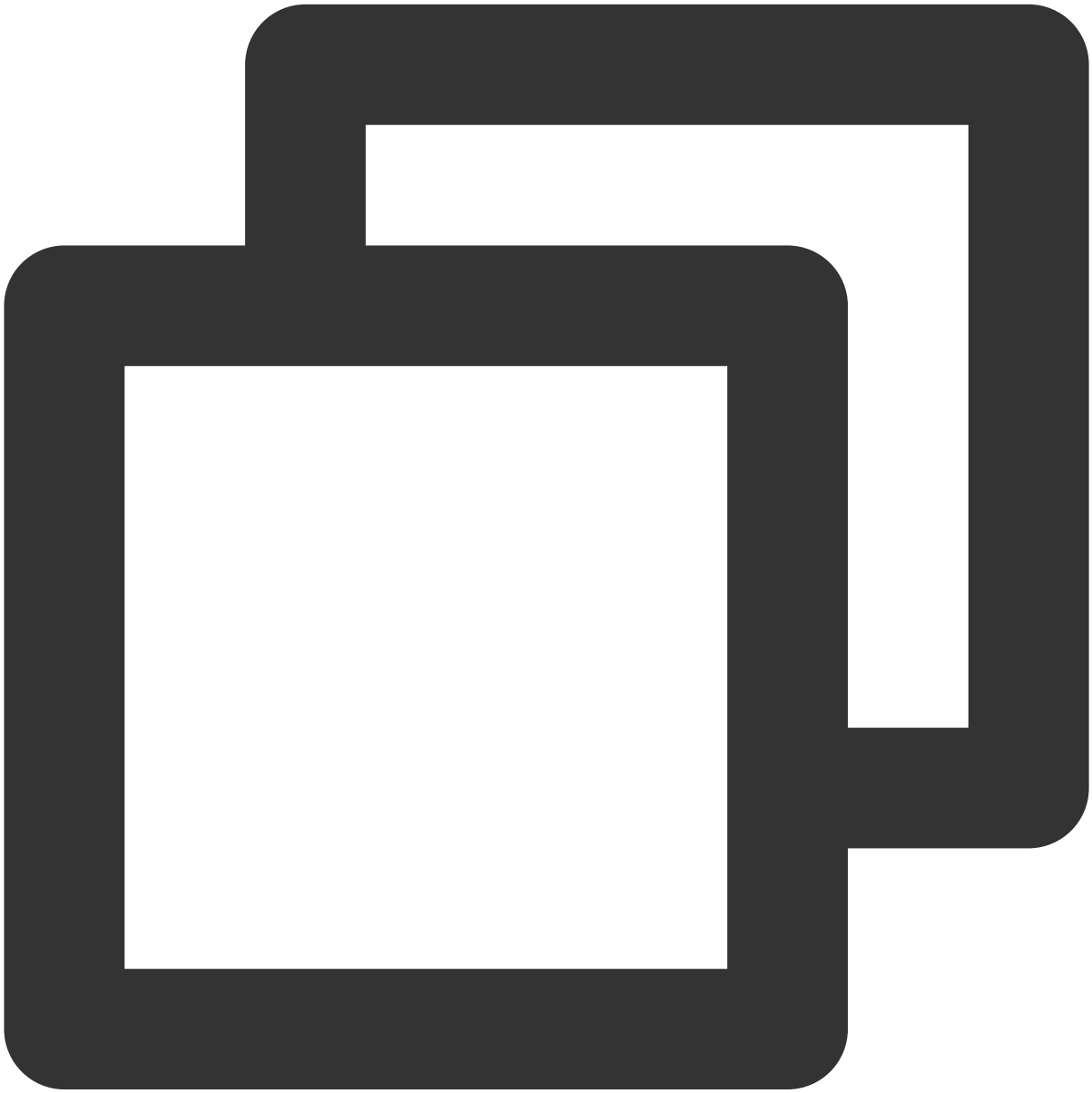
Live video streaming with pictures

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.

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

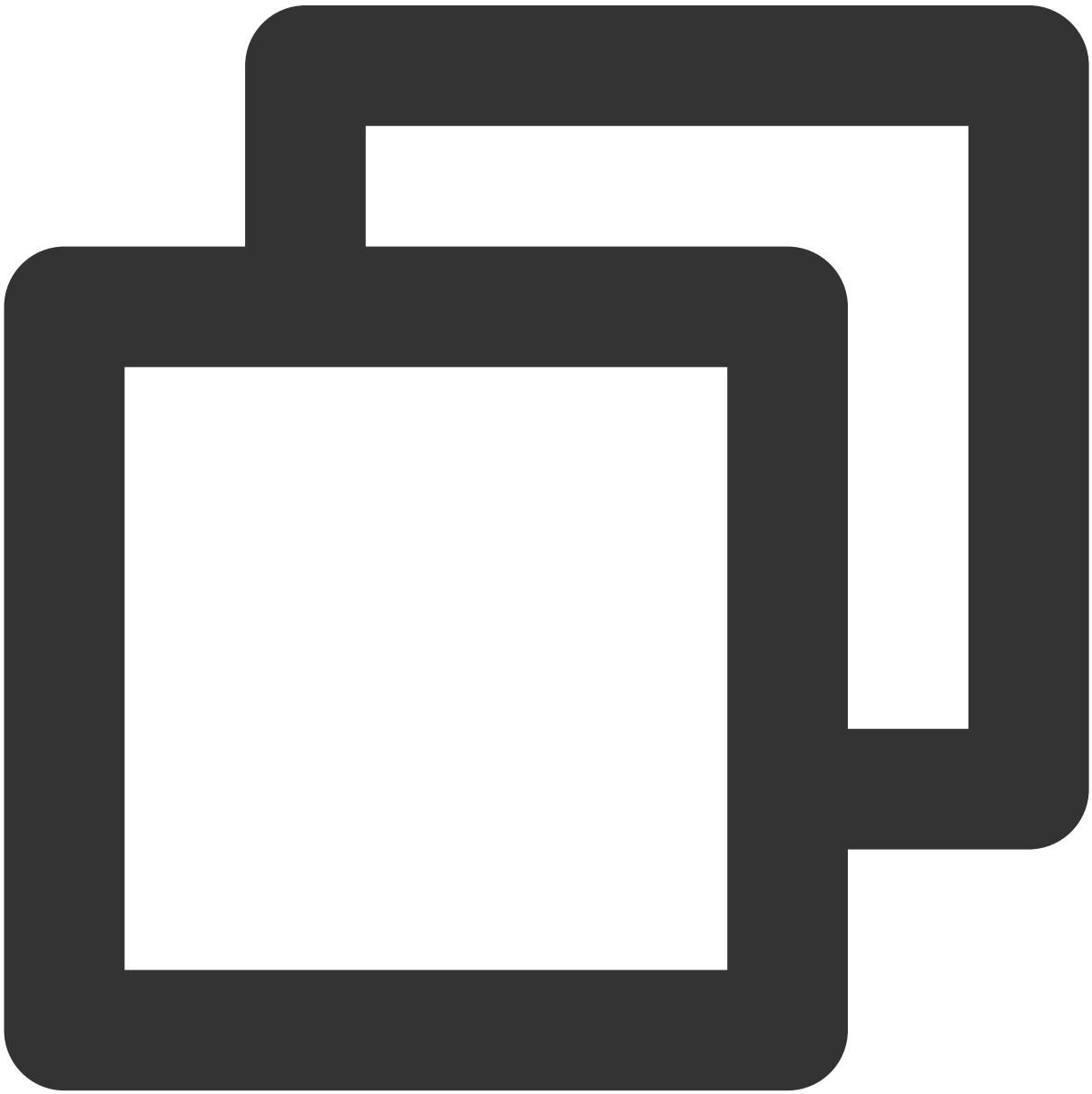


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent">

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

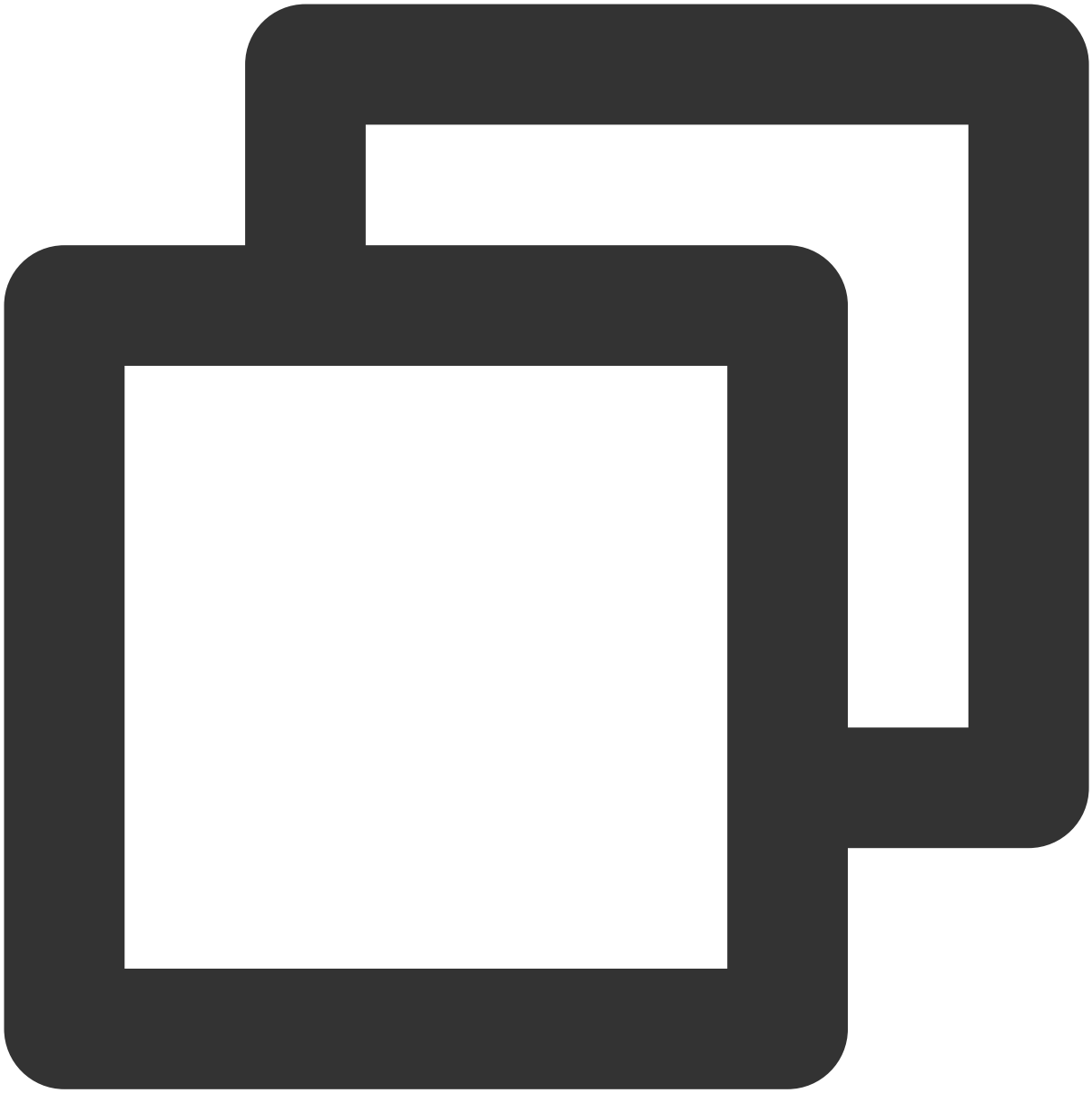
2. Create a new file named `MainActivity.java` and register in the `AndroidManifest.xml` . By loading TUILiveKit `TUILiveListFragment` page, you can Present a list of rooms

Java



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

        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction(
        TUILiveListFragment listFragment = new TUILiveListFragment();
        fragmentTransaction.add(R.id.fl_live_list, listFragment);
        fragmentTransaction.commit();
    }
}
```

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



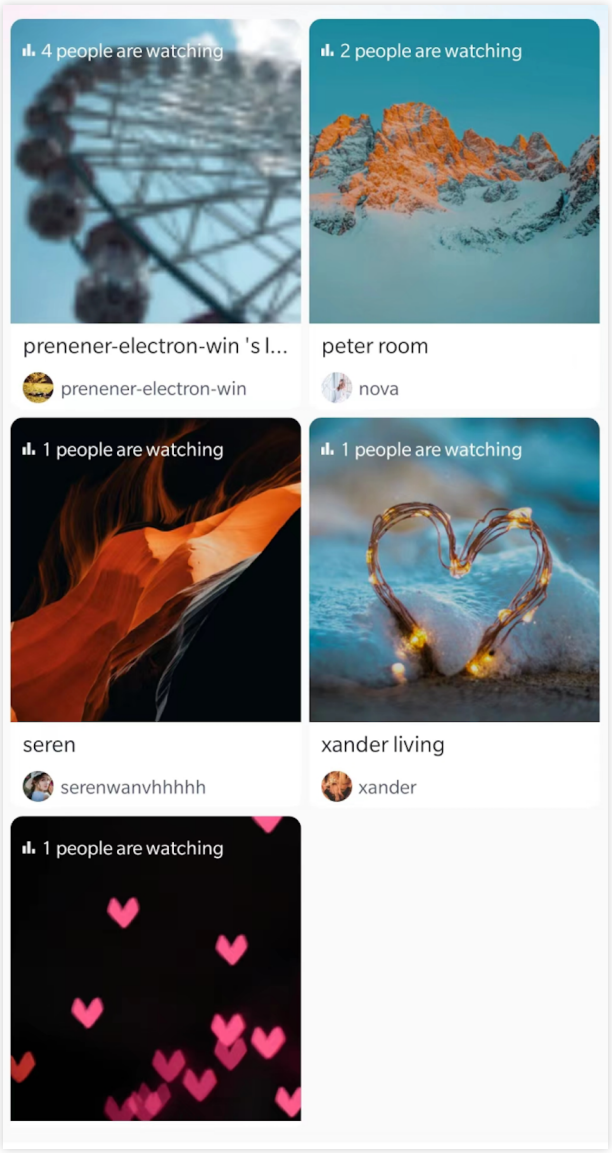
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application>
        ...
        <!-- Example: To register MainActivity, please use your actual package name
        <activity
    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

```
</manifest>
```

Note：

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

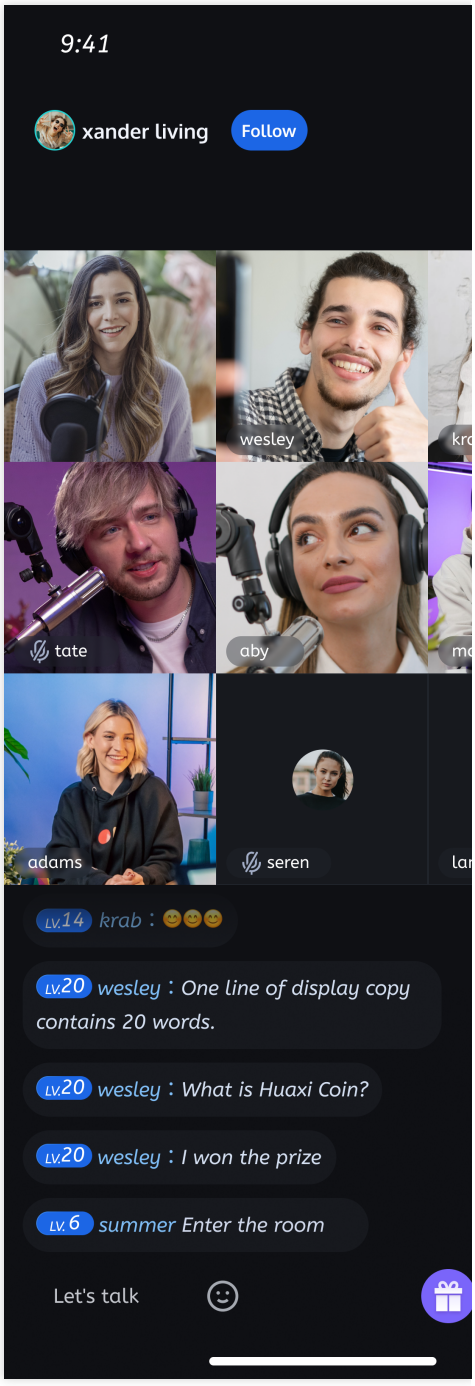
If you encounter problems with Theme.AppCompat, please refer to Activity Theme Questions.



Step 7. The audience enters the studio

On the room list page of Step 6, click any room to automatically enter the live broadcast room.

Video Live Room



Video Live Room

More features

Interactive Bullet Comments

Interactice Gifts

Gift Effects

Beauty Effects

Q&A

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

Suggestions and Feedback

If you have any suggestions or feedback, please contact info_rtc@tencent.com.

Electron

Last updated：2024-07-29 15:34:04

This document will guide you on how to quickly integrate the desktop TUILiveKit component into your project, thereby providing your application with live streaming capabilities.

Environmental Preparation

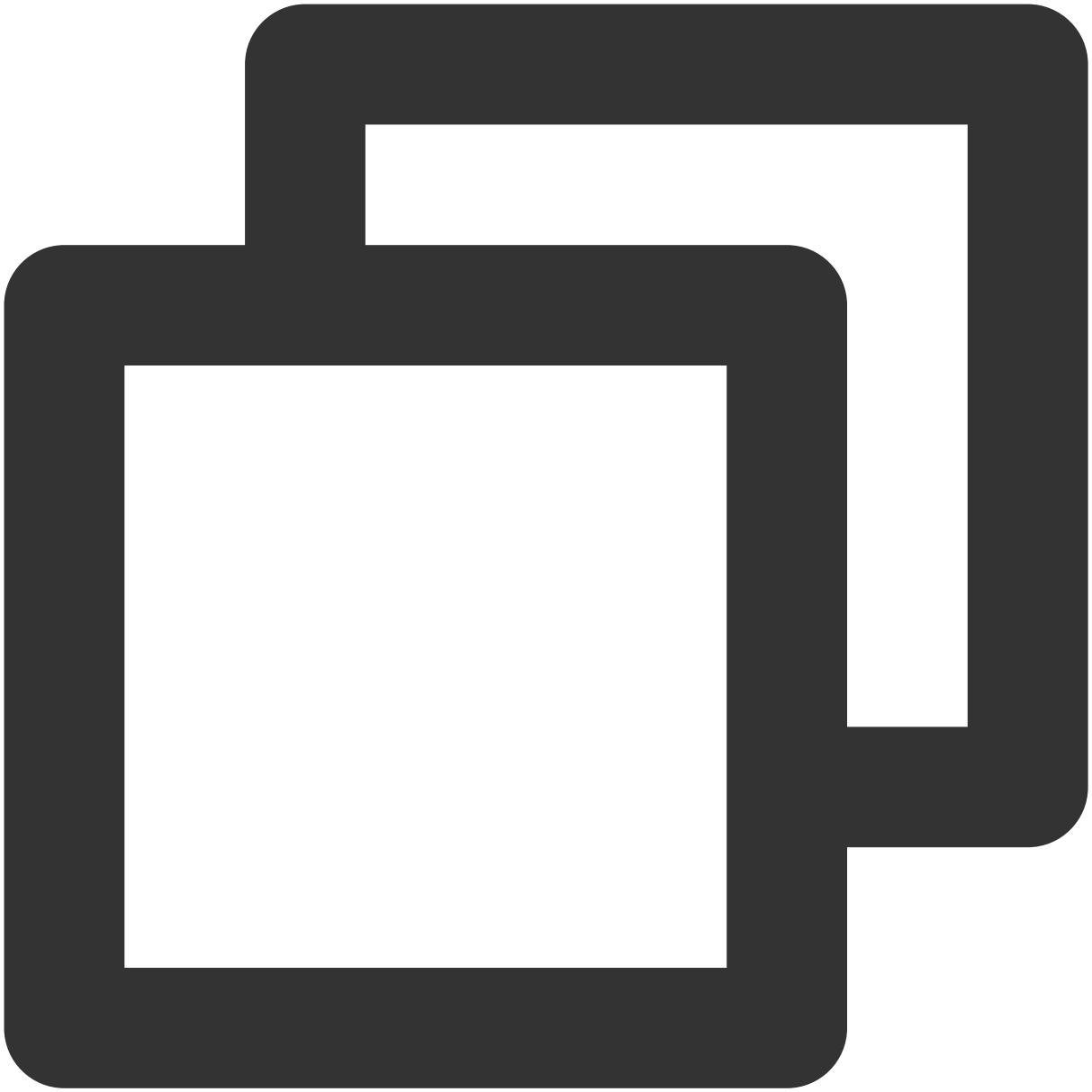
Operating System: Windows 10 or 11.
Node.js version ≥ 16.19.1 (Recommended to use the official LTS version, and the npm version should match the node version).

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 source code

Get open source code from Github, or you can clone the code with the git command as below:



```
git clone https://github.com/Tencent-RTC/ultra-live-electron.git
```

Step 3: Integrate TUILiveKit

When TUILiveKit runs on the desktop, two Electron browser windows need to be created to accommodate the main page view and settings page view, respectively. We call these two windows the TUILiveKit main window and sub-window, respectively. After integrating TUILiveKit into your existing application, you can send a message to Electron

main process, for exapme by cilcking a button, to open TUILiveKit main window. Then you can explore all features of TUILiveKit.

Prerequirement

Your existing code project need to include the following technology support:

Vue3

Webpack

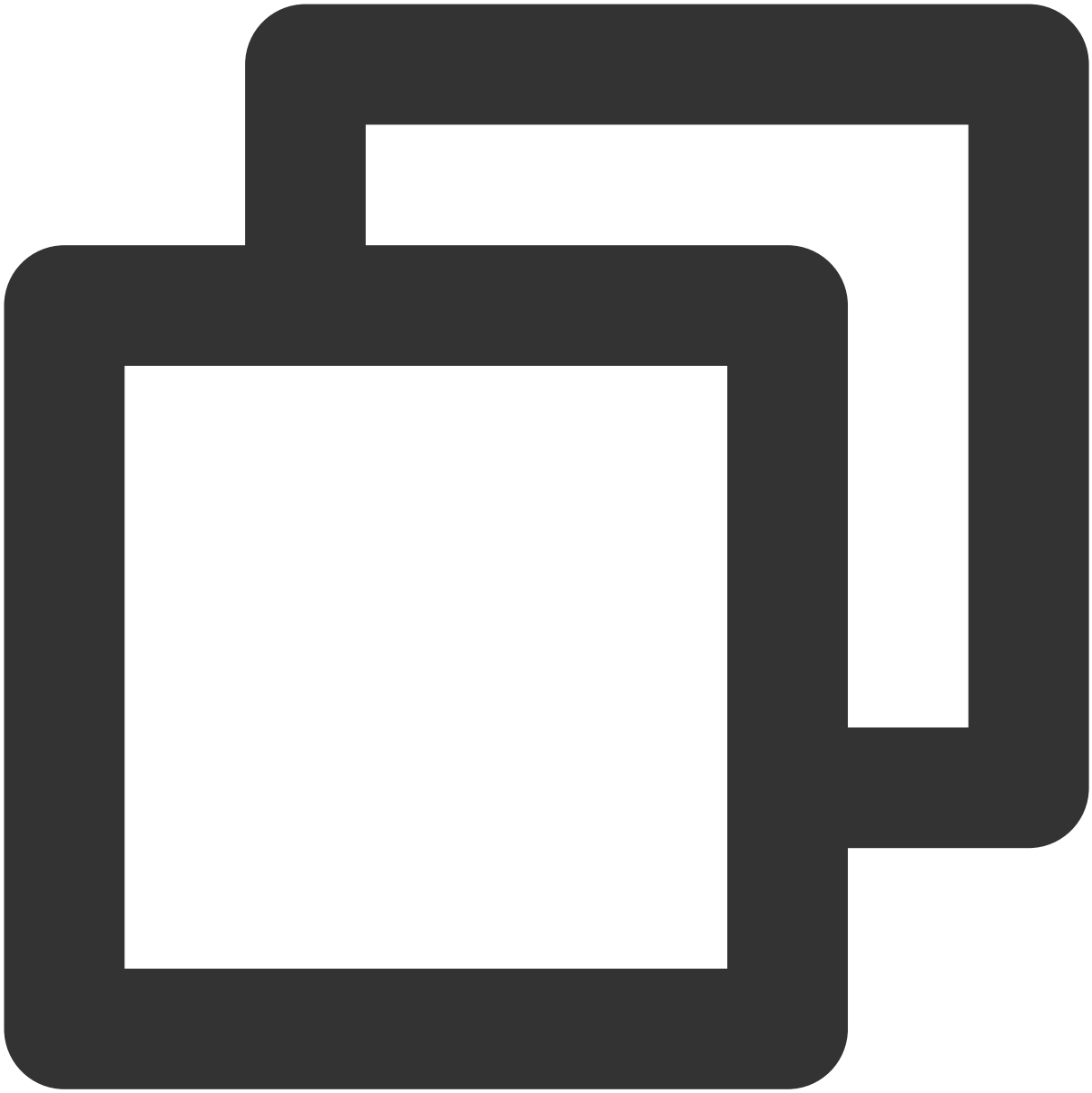
TypeScript

Electron

Note：

If you do not have a project that meets the integration prerequirement, you can refer to the common questionsat the bottom of the document for guidance.

Install dependencies



```
npm install --save pinia
npm install --save trtc-electron-sdk@11.8.603-alpha.0
npm install --save @tencentcloud/tuiroom-engine-electron@2.4.0-alpha.2
npm install --save trtc-electron-plugin-xmagic@latest
npm install --save-dev native-ext-loader electron-devtools-installer electron-build
```

Copy TUILiveKit soure code to your project

1. **Copy TUILiveKit components**

Copy directory `ultra-live-electron/src/TUILiveKit` into `src` directory of your project.

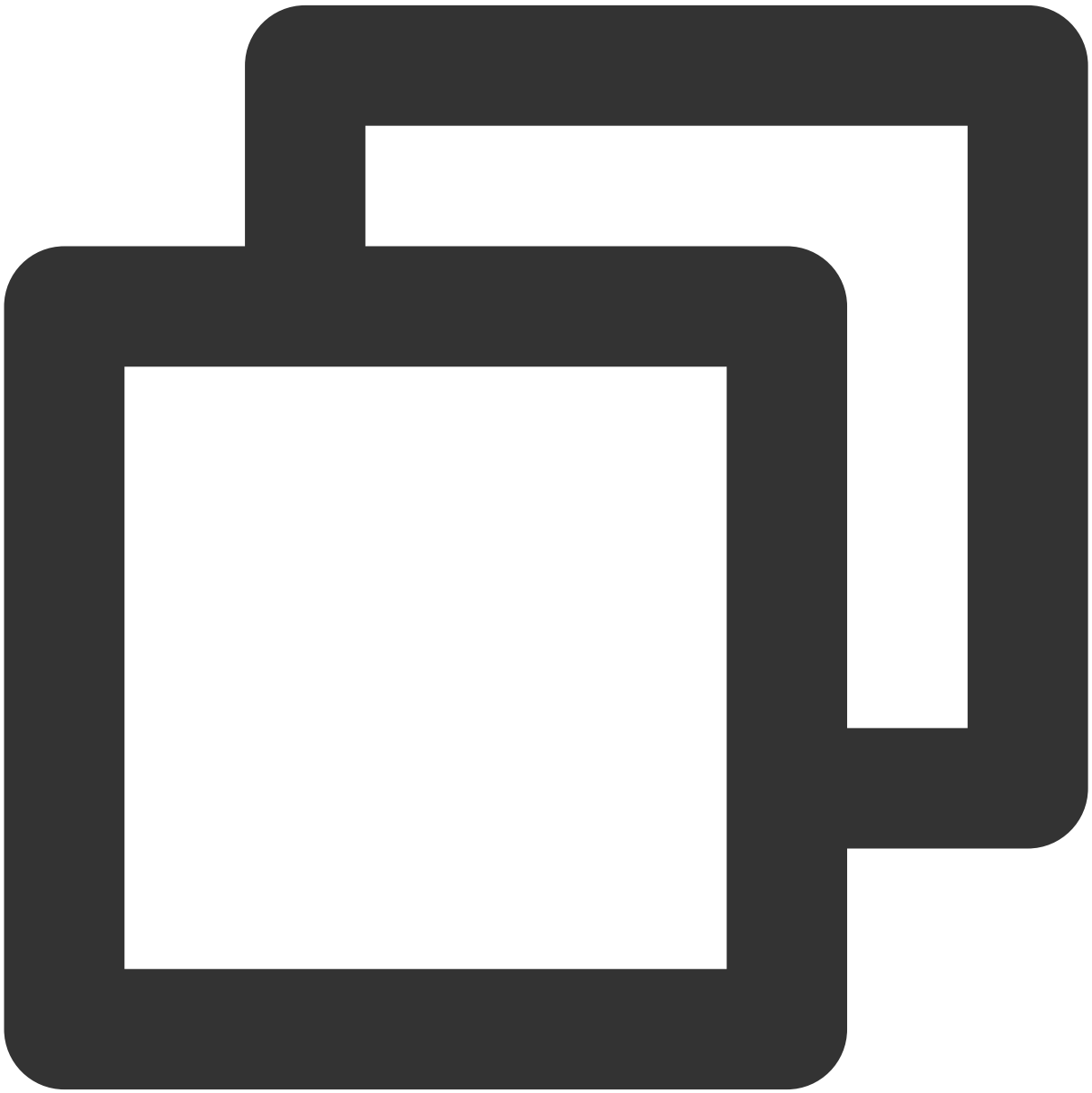
2. **Copy TUILiveKit windows creation code**

Copy `ultra-live-electron/TUILiveKit.main.js` and `ultra-live-electron/TUILiveKit.preload.js` files into root directory of your project.

3. **Copy TUILiveKit main window and sub-window pages and their router configuration**

Copy `ultra-live-electron/src/views/TUILiveKitChild.vue` and `ultra-live-electron/src/views/TUILiveKitMain.vue` files into `src/views` directory of you project。

Add the following pages routing configuration in `src/router/index.ts` file of your project:



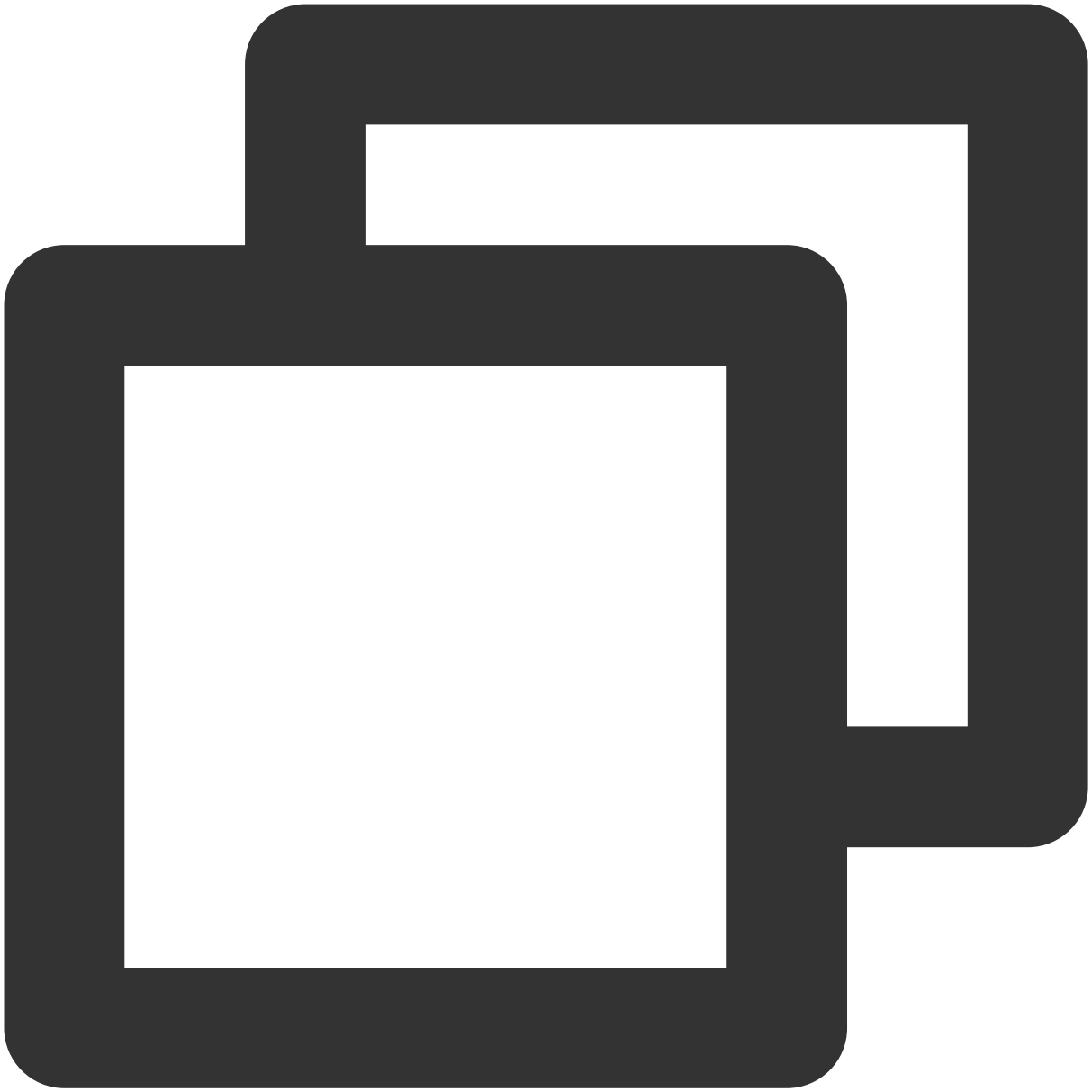
```
// src/router/index.ts
import { createRouter, createWebHashHistory, RouteRecordRaw } from 'vue-router';
```

```
import HomeView from '../views/HomeView.vue';

const routes: Array<RouteRecordRaw> = [
  { // Default page of your application, contains a button to trigger the openning
    path: '/',
    name: 'home',
    component: HomeView
  },
  // ********************************* TUILiveKit required code start ************
  { // TUILiveKit main window page
    path: '/tui-live-kit-main',
    name: 'tui-live-kit-main',
    component: () => import(/* webpackChunkName: "TUILiveKitMain" */ '../views/TUIL
  },
  { // TUILiveKit sub-window page
    path: '/tui-live-kit-child',
    name: 'tui-live-kit-child',
    component: () => import(/* webpackChunkName: "TUILiveKitChild" */ '../views/TUI
  },
  // ********************************* TUILiveKit required code end **************
];

// ********************************* TUILiveKit required code start **************
window.ipcRenderer.on('window-type', (event: any, type: string) => {
  console.log(`[router] window type:${type}`);
  console.log(`[router] current href:${window.location.href}`);
  router.replace({ name: `tui-live-kit-${type}` });
});
// ********************************* TUILiveKit required code end ****************

const router = createRouter({
  history: createWebHashHistory(),
  routes
});

export default router;
```

4. **Copy beauty-related code, resources, and configurations.**

Copy `ultra-live-electron/public/assets` directory and `ultra-live-electron/public/avatar.png` file into the `public` directory of your project.

Copy `ultra-live-electron/scripts/prestart.js` file into the `scripts` directory of your project, and add the following command in the `scripts` section of the `package.json` file of your project.

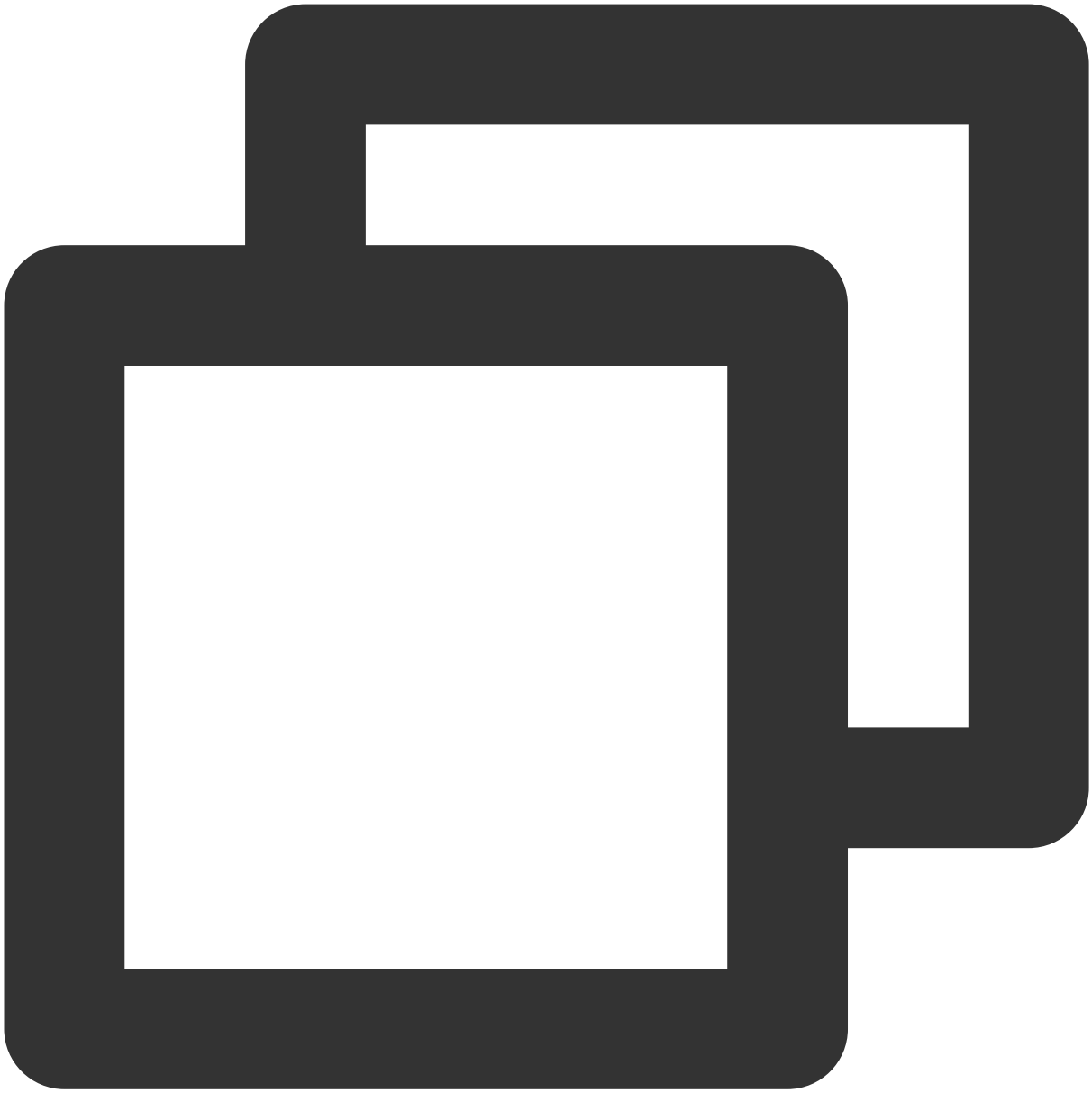


```
{
  "scripts": {
    "prestart": "node ./scripts/prestart.js"
  }
}
```

Here we do not enable the humain beauty function right now. The above configuration, code, and resource copied can ensure the project runs without errors. For how to enable the beauty function, see How to Enable the Beauty Function.

5. **Modify vue.config.js**

Add the following configuration in the `vue.config.js` file of your project:

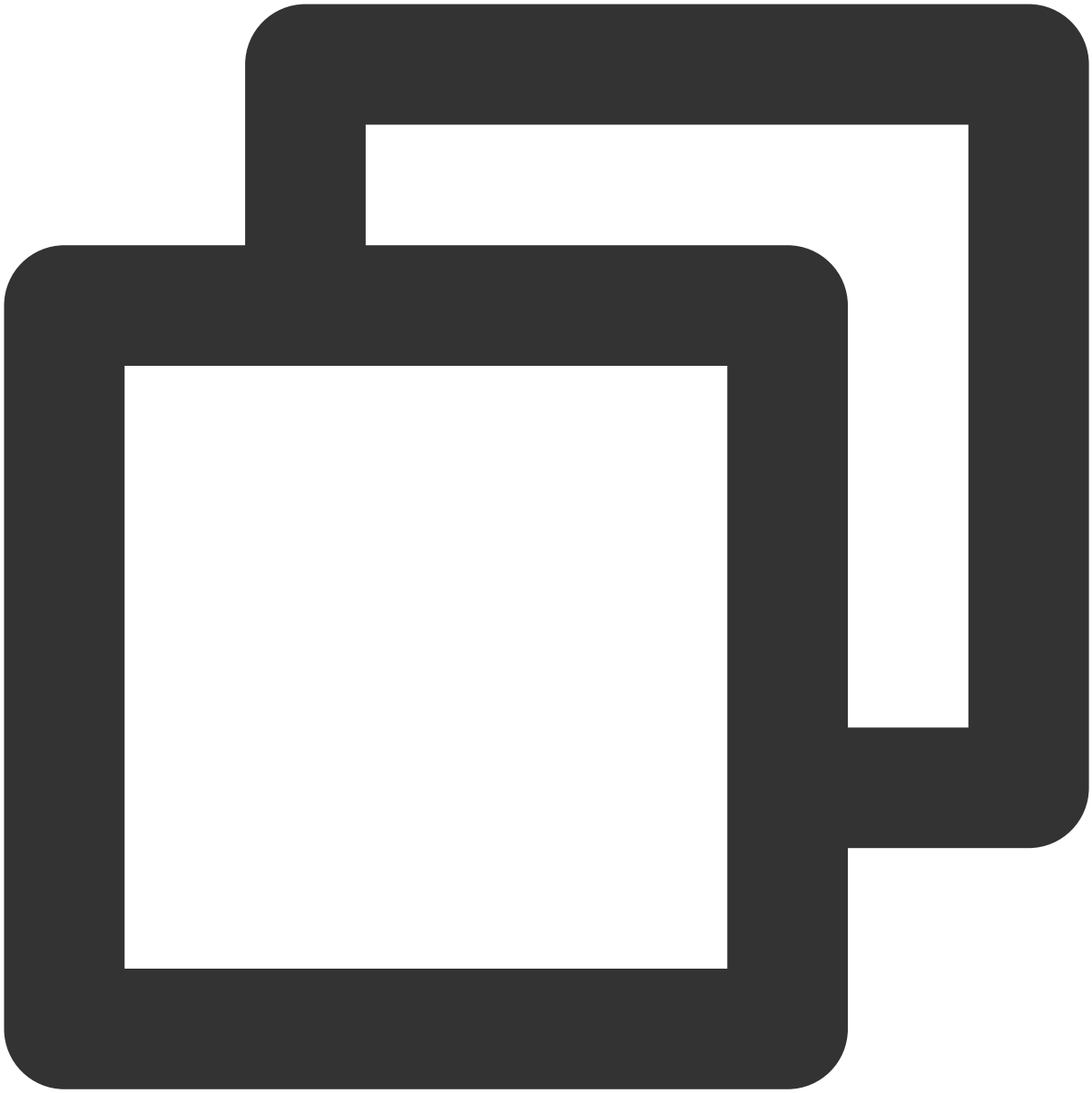


```
// vue.config.js
const { defineConfig } = require('@vue/cli-service')
// ********************************** TUILiveKit required code start *************
const os = require("os");
const isProduction = process.env.NODE_ENV === "production";
const platform = os.platform();
// ********************************** TUILiveKit required code end ***************

module.exports = defineConfig({
  transpileDependencies: true,
// ********************************** TUILiveKit required code start *************
```

```
    publicPath: "./",
    configureWebpack: {
      devtool: isProduction ? "source-map" : "inline-source-map",
      target: "electron-renderer",
      module: {
        rules: [
          {
            test: /\\.node$/,
            loader: "native-ext-loader",
            options: {
              rewritePath: isProduction
                ? platform === "win32"
                  ? "./resources"
                  : "../Resources"
                : "./node_modules/trtc-electron-sdk/build/Release",
            },
          },
        ],
      },
    }
  // ********************************** TUILiveKit required code end ***************
  });
```

6. Copy `src/debug` directory and configure `SDKAppID` and `SDKSecretKey` .

Copy `ultra-live-electron/src/debug` directory into the `src` directory of your project, open the copied file `basic-info-config.js` to enter the `SDKAppID` and `SDKSecretKey` obtained from Step 1: Activate the service .

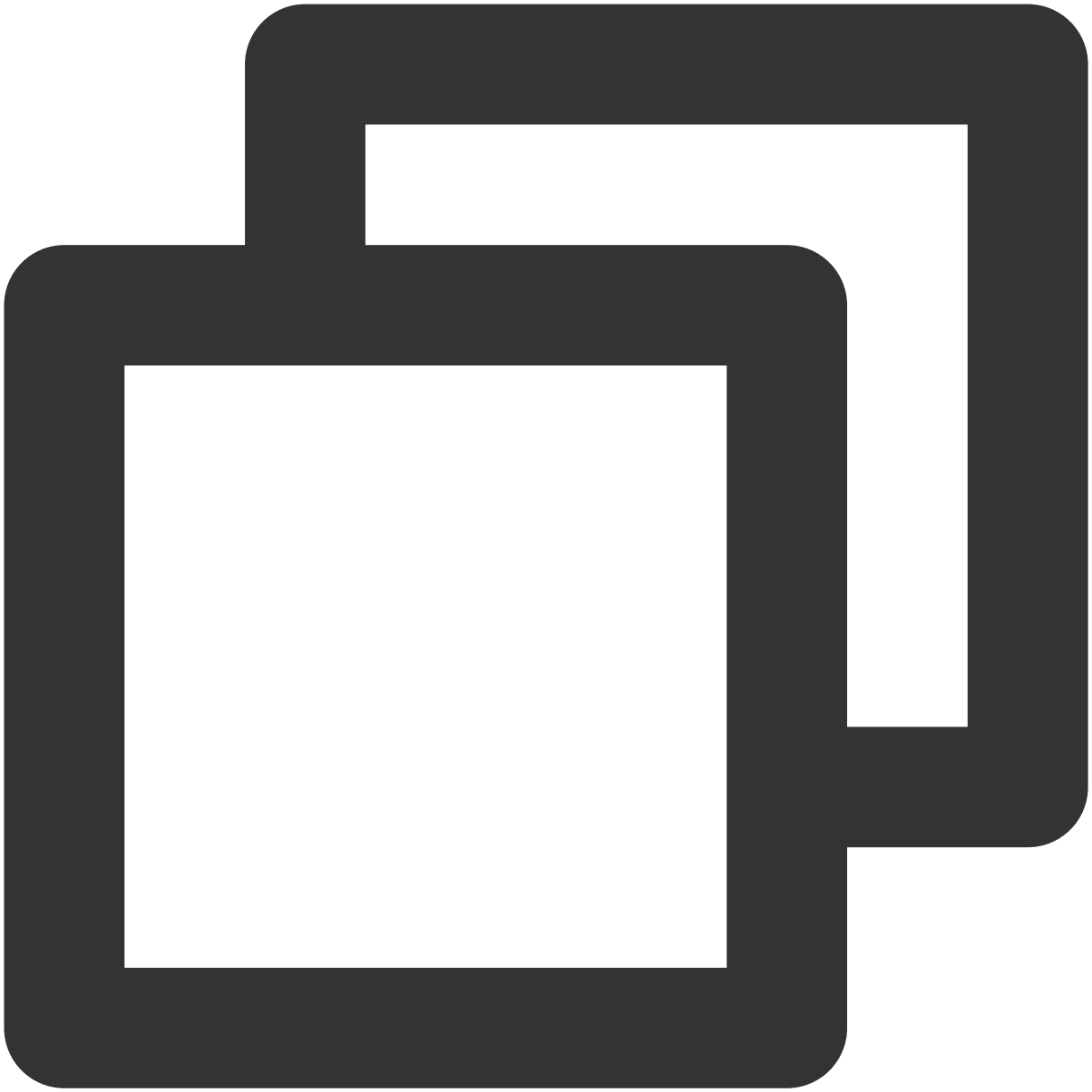


```
// basic-info-config.js
/**
 * Tencent Cloud SDKAppID, which should be replaced with user's SDKAppID.
 * Enter Tencent Cloud TRTC [Console] (https://console.tencentcloud.com/trtc ) to c
 * and you will see the SDKAppID.
 * It is a unique identifier used by Tencent Cloud to identify users.
 */
export const SDKAppID = 0;


/**
```

```
  * Encryption key for calculating signature, which can be obtained in the following
  *
  * Step1. Enter Tencent Cloud TRTC [Console](https://console.tencentcloud.com/rav )
  * and create an application if you don't have one.
  * Step2. Click your application to find "Quick Start".
  * Step3. Click "View Secret Key" to see the encryption key for calculating UserSig
  * and copy it to the following variable.
  *
  * Notes: this method is only applicable for debugging Demo. Before official launch
  * please migrate the UserSig calculation code and key to your backend server to av
  * unauthorized traffic use caused by the leakage of encryption key.
  * Document: https://www.tencentcloud.com/document/product/647/35166#Server
  */
 export const SDKSecretKey = '';
```

7. Enable pinia

Open the `src/main.ts` file in your project, add the following configuration to enable pinia:

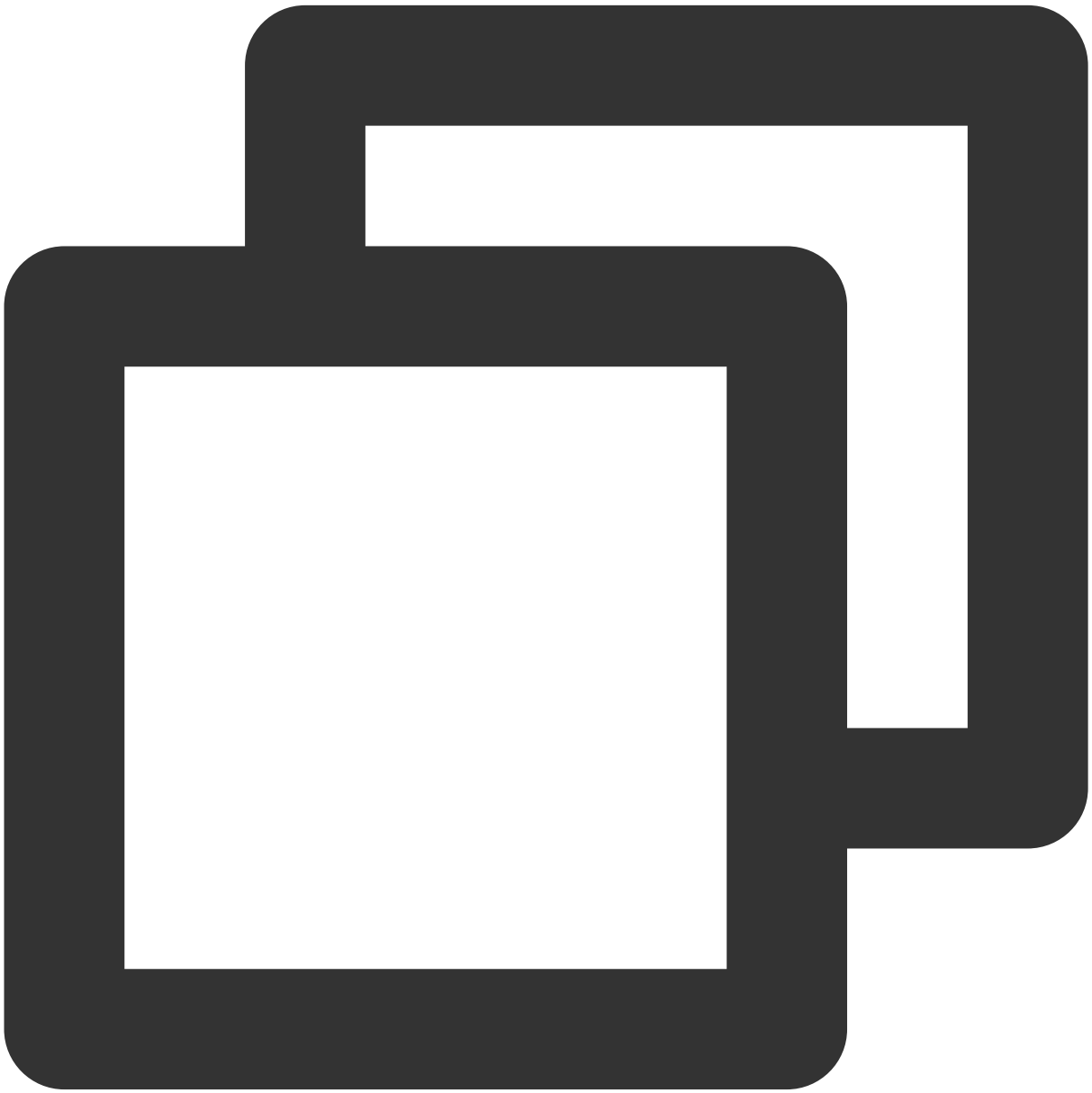


```
// main.ts
import { createApp } from 'vue';
// ********************************** TUILiveKit required code start *************
import { createPinia } from 'pinia';
// ********************************** TUILiveKit required code end ***************
import App from './App.vue'
import router from './router'

createApp(App)
// ********************************** TUILiveKit required code start *************
  .use(createPinia())
```

```
// ********************************** TUILiveKit required code end ***************
  .use(router)
  .mount('#app');
```

8. **Modify global.d.ts**

Add the following configuration in your project `src/global.d.ts` file. Need to declare several properties on the global Window type：



```
// src/global.d.ts
export {}
declare global {
```

```
  interface Window {
// ********************************** TUILiveKit required code start *************
    ipcRenderer: any;
    nativeWindowHandle: Uint8Array;
    mainWindowPort: MessagePort | null;
  }
// ********************************** TUILiveKit required code end ***************
}
```

Add an entry to open the TUILiveKit main window

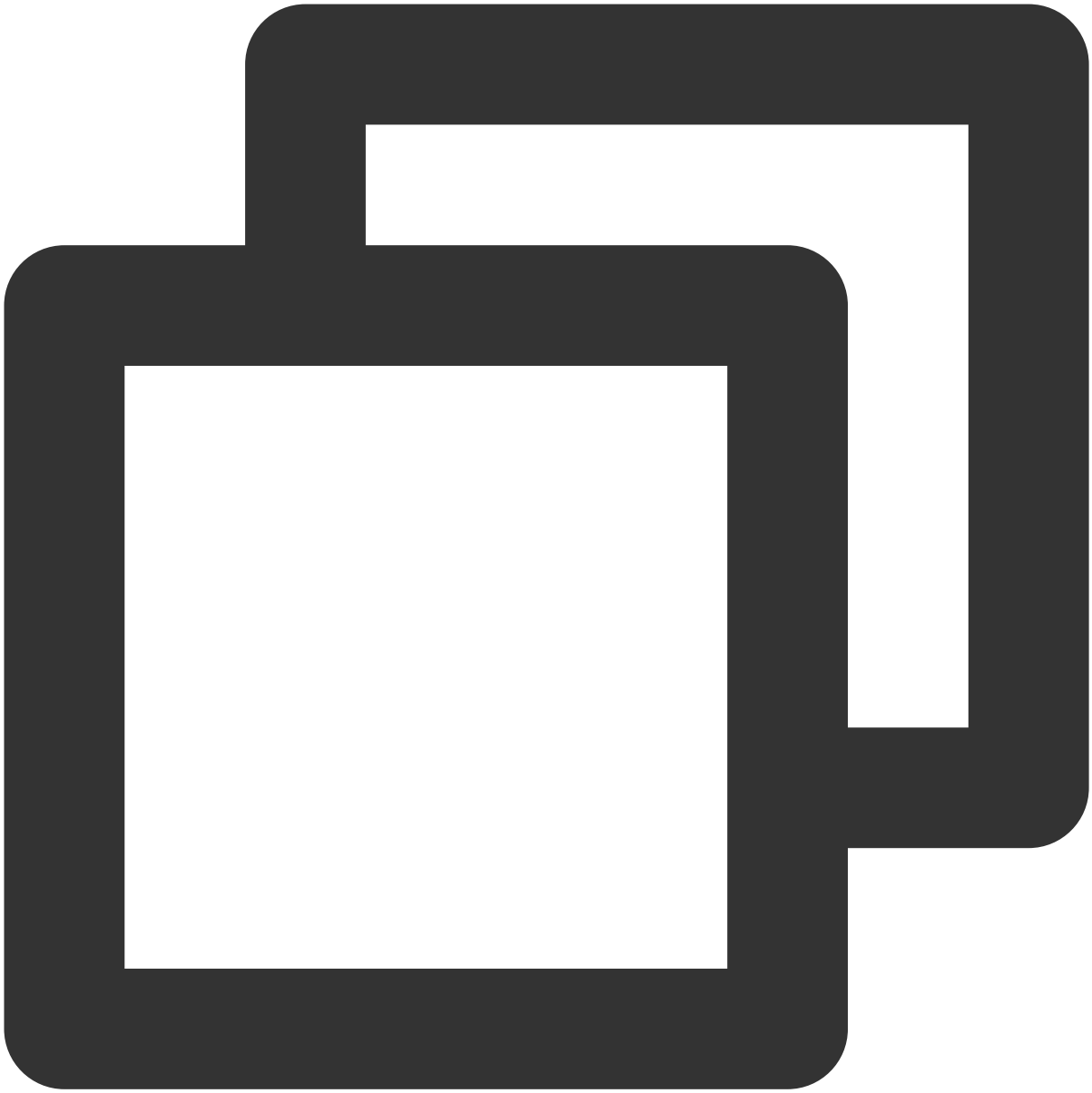
Vue Component

Electron preload script

Eletron main process

package.json

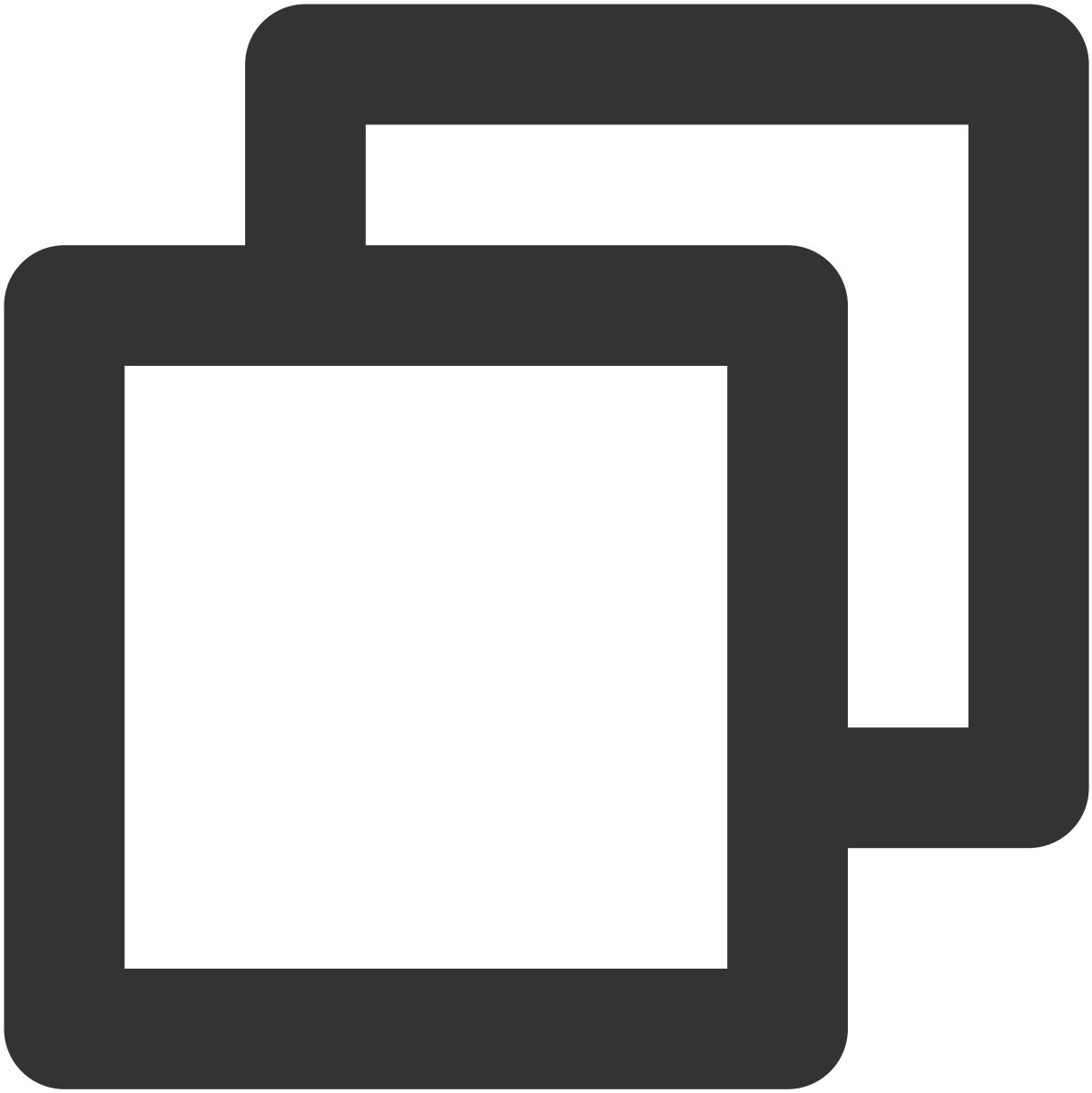
In a page view of your vue project, add a button. When clicked, it will notify the Electron main process to open the TUILiveKit main window. As shown below, this is our implementation example code.



```
// HomeView.vue
<template>
  <div class="home">
    <button @click="openTUILiveKit">Open TUILiveKit</button>
  </div>
</template>

<script setup lang="ts">
import { ref } from 'vue';
import type { Ref } from 'vue';
import { getBasicInfo } from '../debug/basic-info-config';
```

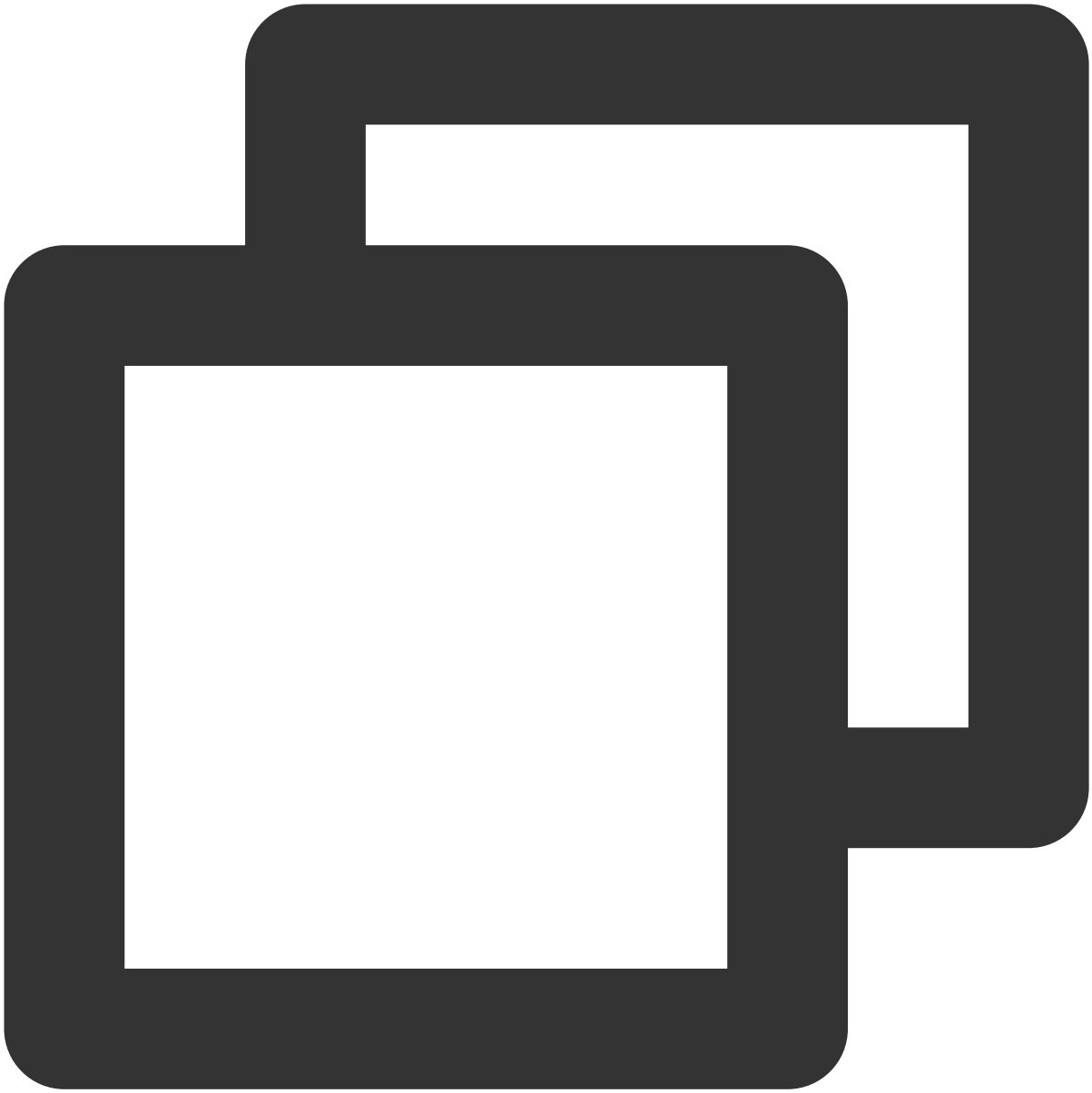
```
const isOpen:Ref<boolean> = ref(false);

const openTUILiveKit = async () => {
  if (!isOpen.value) {
    const currentUserInfo = await getBasicInfo();
    if (currentUserInfo) {
      window.ipcRenderer.send('openTUILiveKit', {
        userInfo: currentUserInfo // Note: User information is required to open TUI
      });
      isOpen.value = true;
    } else {
      console.error('Error: cannot get current user info');
    }
  }
};
</script>
```

To facilitate communication with the Electron main process in Vue components and JavaScript/TypeScript scripts, put the ipcRenderer object of Electron onto the global object window in the preload script of your Electron project.



```
// preload.js
const { ipcRenderer } = require("electron");

// Enable `ipcRenderer` can be used in vue and Javascript module
window.ipcRenderer = ipcRenderer;
```

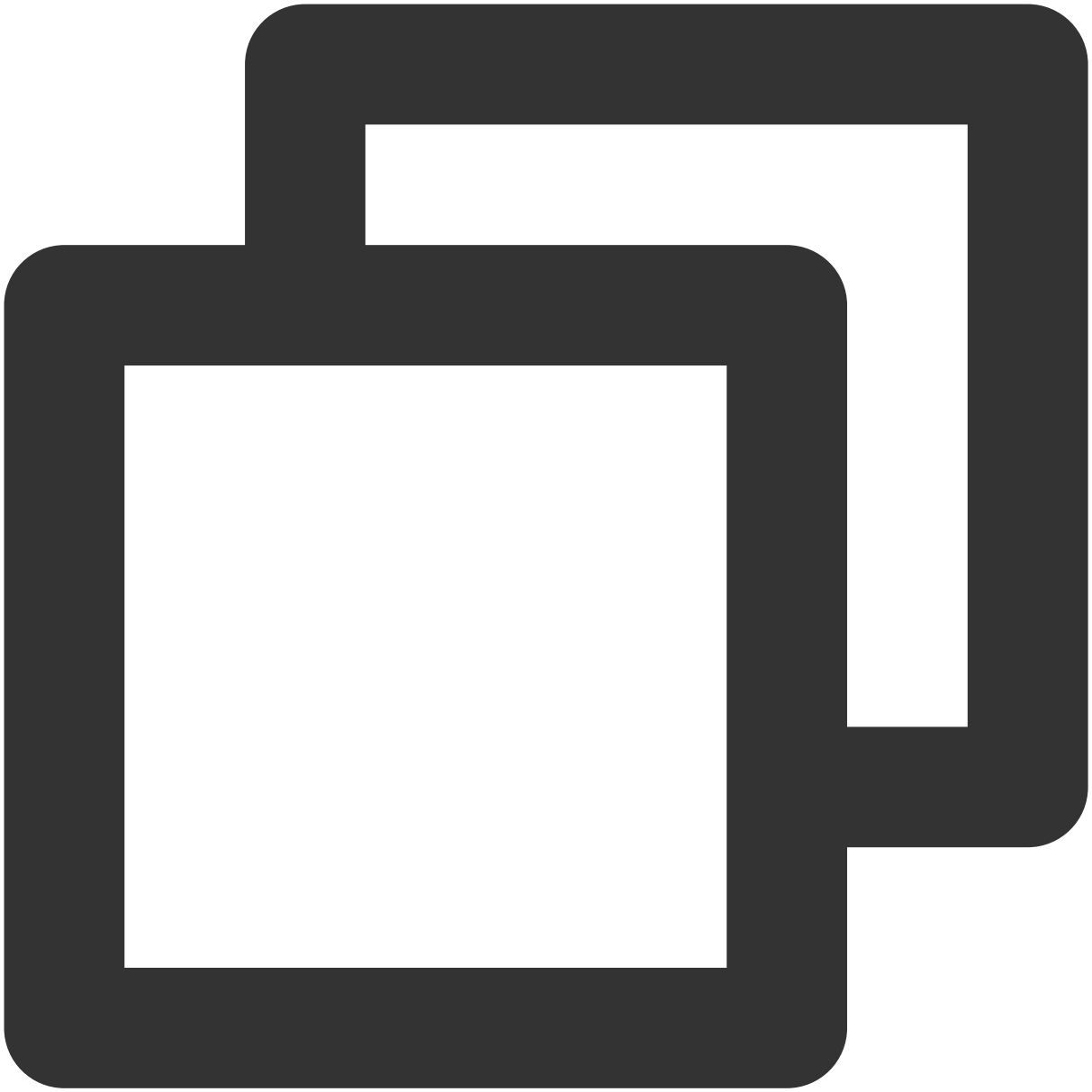
When received `openTUILiveKit` message from Vue component, the Electron main process will open the TUILiveKit main window.



```
// main.js
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');
// ********************************** TUILiveKit required code start *************
const { TUILiveKitMain } = require("./TUILiveKit.main");
// ********************************** TUILiveKit required code end ***************

let mainWindow = null;
const createWindow = () => {
  mainWindow = new BrowserWindow({
    width: 800,
```

```
    height: 600,
// ********************************** TUILiveKit required code start *************
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: true,
      contextIsolation: false,
    }
// ********************************** TUILiveKit required code end ***************
  });

// ********************************** TUILiveKit required code start *************
  bindIPCMainEvent();
// ********************************** TUILiveKit required code end ***************

  if (app.isPackaged) {
    mainWindow.loadFile("dist/index.html");
  } else {
    mainWindow.loadURL('http://localhost:8080');
  }
}

// ********************************** TUILiveKit required code start *************
function bindIPCMainEvent() {
  ipcMain.on("openTUILiveKit", (event, args) => {
    console.log(`[main] open live kit`, args);
    TUILiveKitMain.open(args); // Open TUILiveKit main window
  });
}
// ********************************** TUILiveKit required code end ***************


app.whenReady().then(() => {
  createWindow();
  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  });
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
});
```

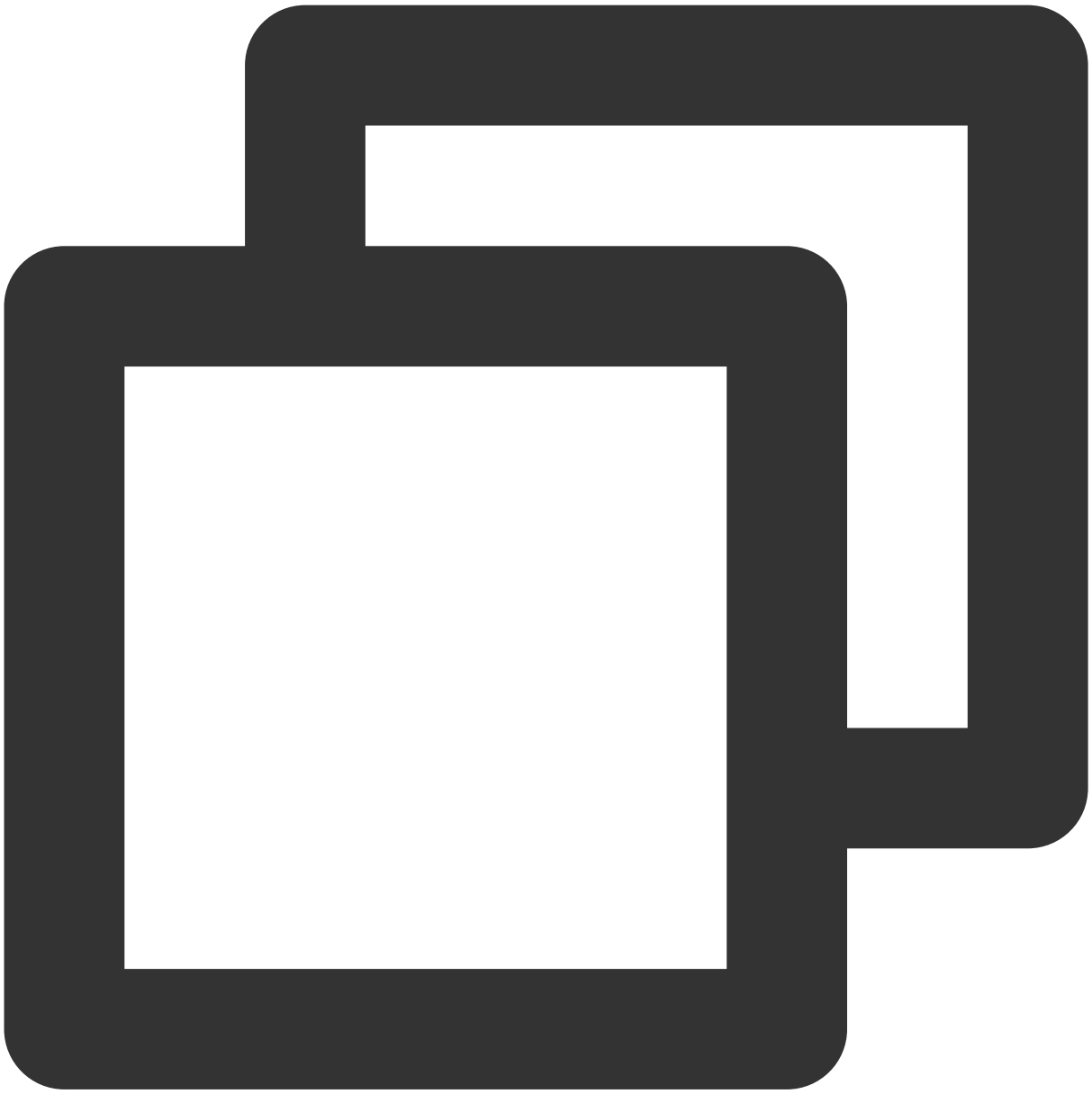
Add the following command in "scripts" of your project package.json file to enable the starting of Electron applicaiton in development mode.



```
{
  "scripts": {
      "start": "electron ."
  }
}
```

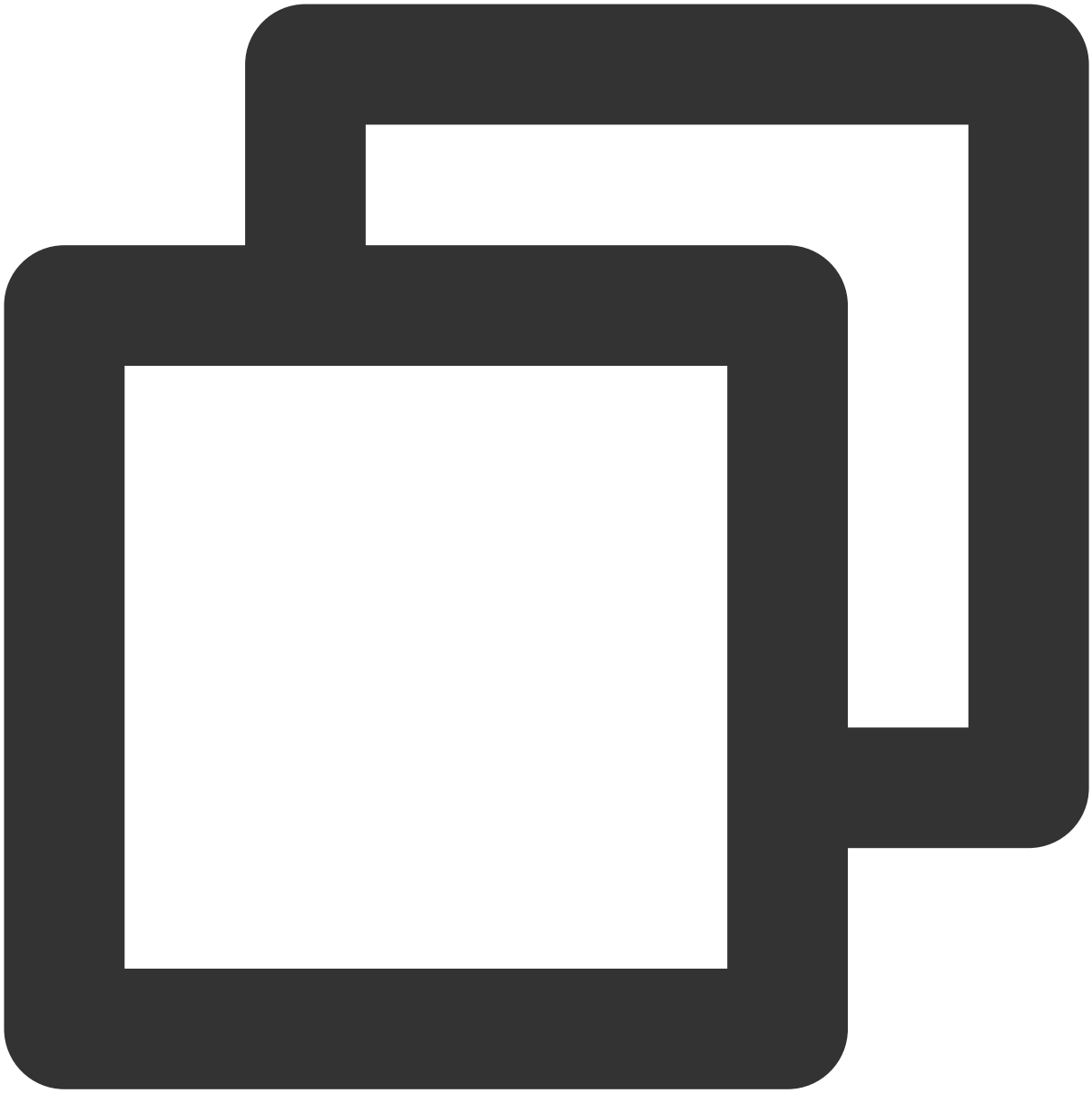
Step 4: Run in development mode

1. Enter the root diectory of your project with `cmd.exe` and execute the following command to start the vue web project.



```
npm run serve
```

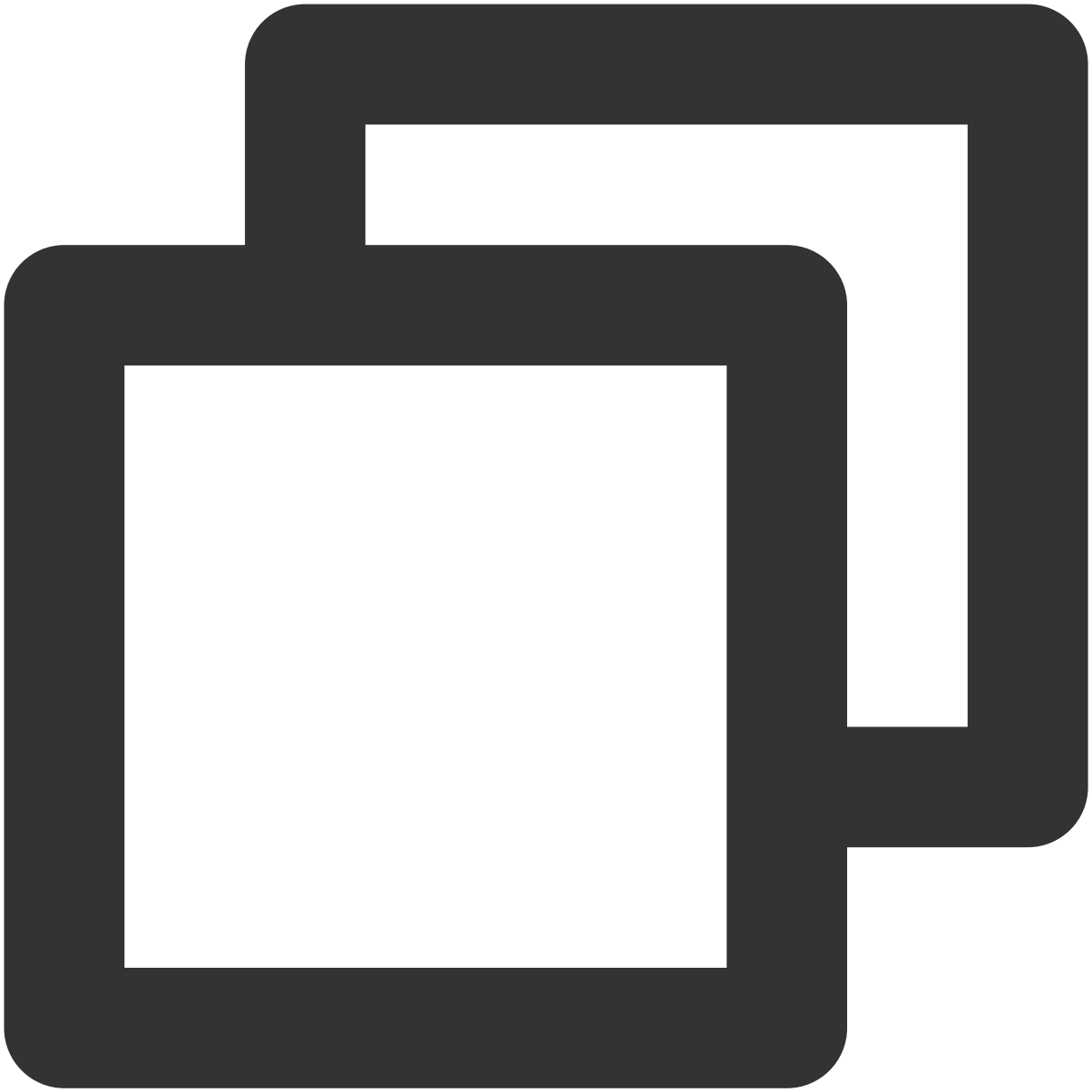
If you encounter an **error Component name "Index" should always be multi-word vue/multi-word-component-names** at startup, it indicates that there is a difference in eslint configuration between your project and TUILiveKit. Add the following "**vue/multi-word-component-names**" validation rule in your project's **.eslintrc.js** file or **eslintConfig** section of the **package.json** file.



```
// .eslintrc.js
module.exports = {
  root: true,
  env: {
    node: true
  },
  'extends': [
    'plugin:vue/vue3-essential',
    'eslint:recommended',
    '@vue/typescript/recommended'
  ],
```

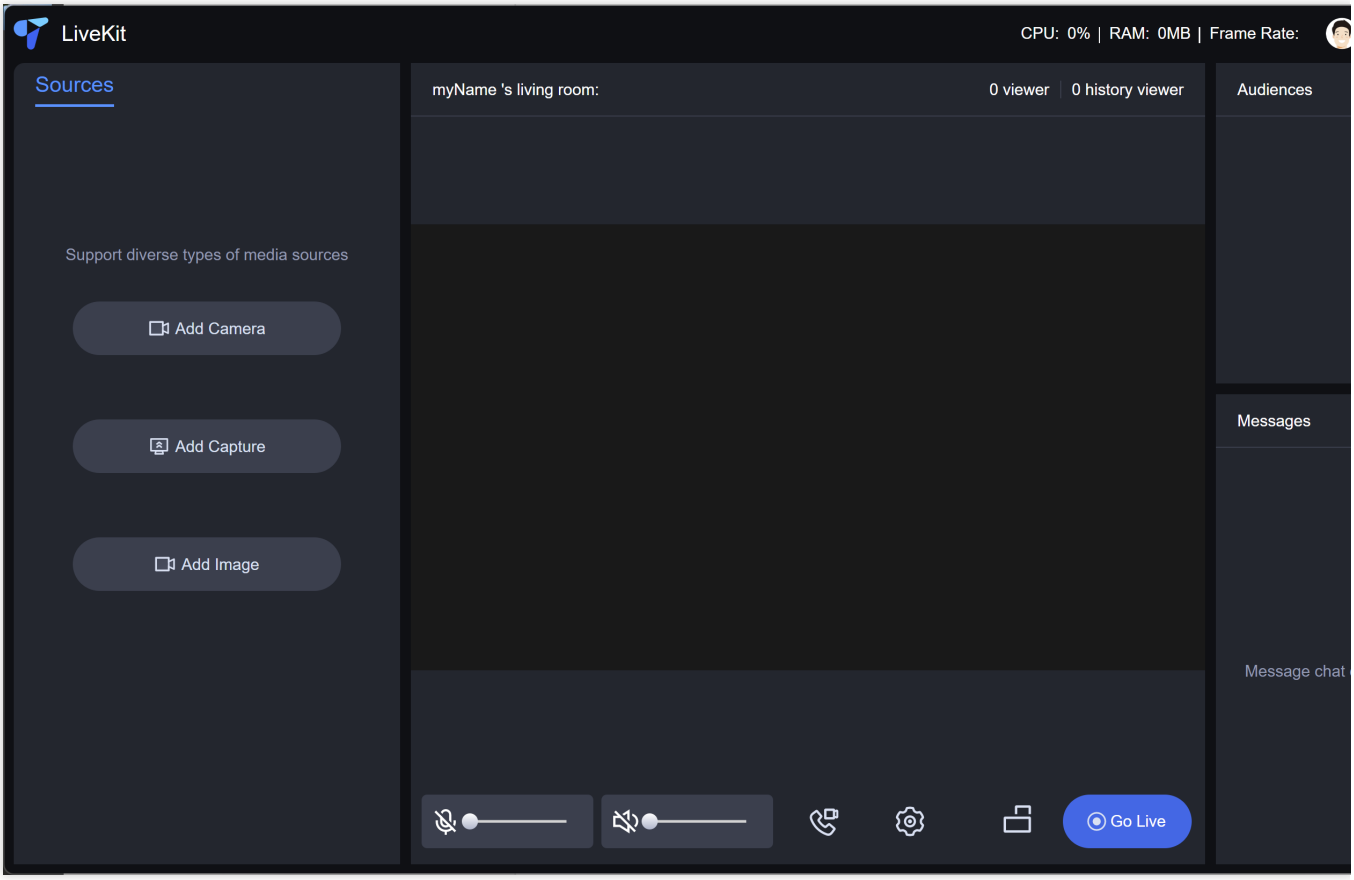
```
  parserOptions: {
    ecmaVersion: 2020
  },
  rules: {
    'no-console': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
    'no-debugger': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
// ********************************** TUILiveKit required code start *************
    "vue/multi-word-component-names": process.env.NODE_ENV === 'production' ? 'warn
// ********************************** TUILiveKit required code end ***************
  }
}
```

2. Enter the root directory of your project with **another** `cmd.exe` and execute the following command to start Electron applicaiton in development mode:



```
npm run start
```

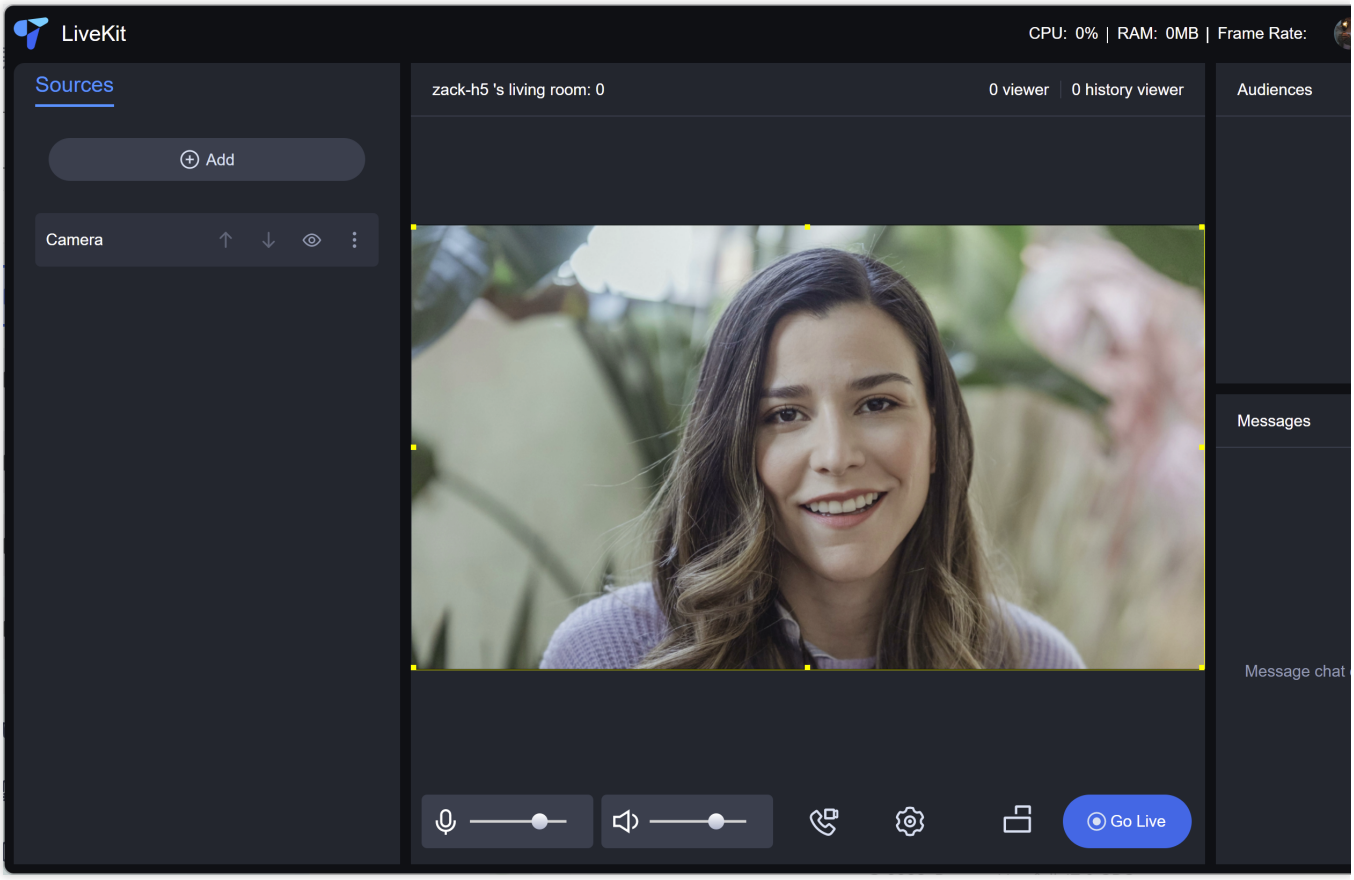
After successfully started, Click the "Open TUILiveKit" button to open TUILiveKit main window which is showing as below:



Step 5: Start your first live broadcast

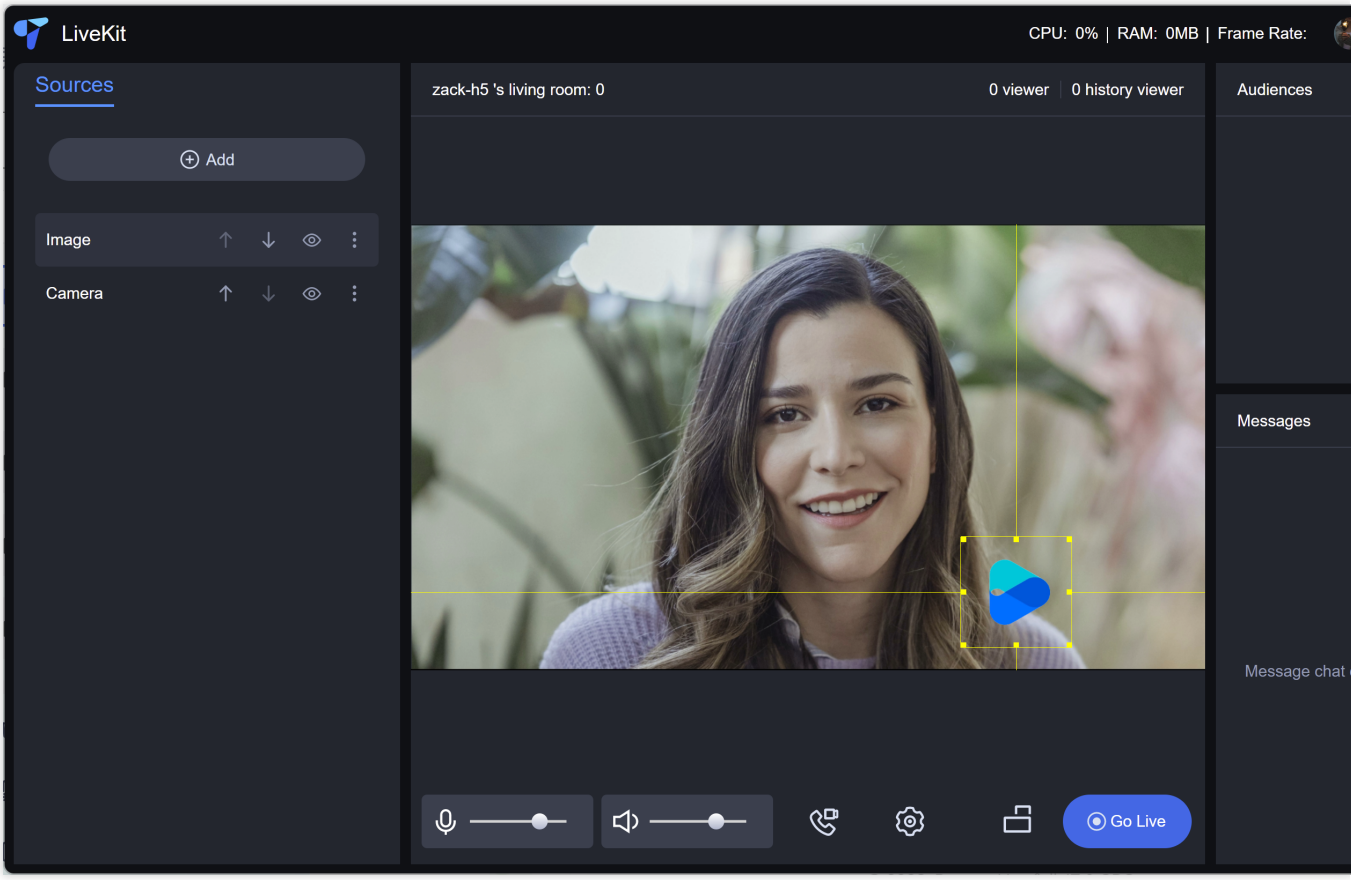
1. **Add a camera**

Firstly, you should add some multimedia source before start live broadcast. Multimedia sources supported include: camera, image, screen and window capture.For example, the image below shows the effect after adding a camera.



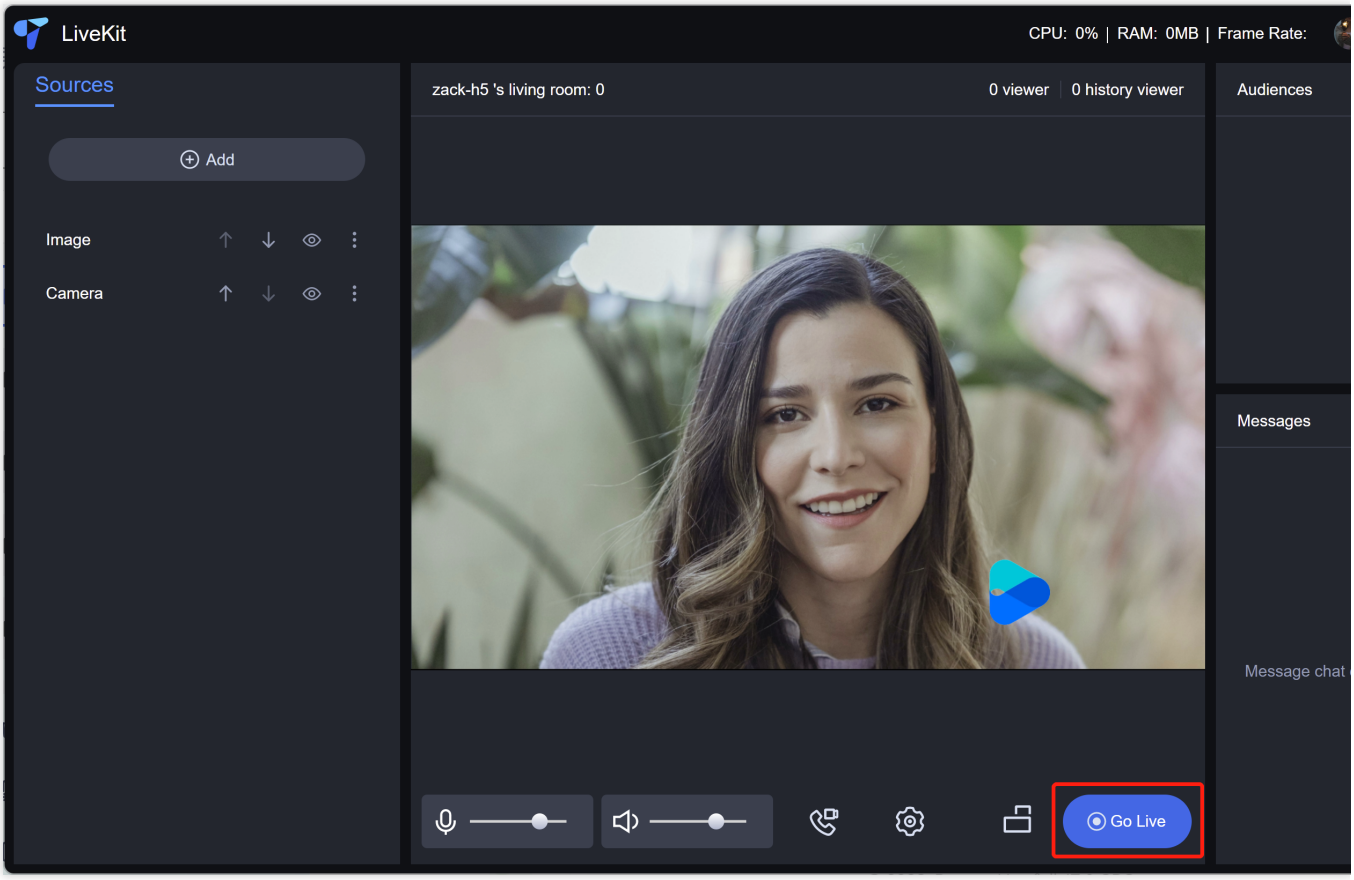
2. **Add your logo image**

If you need to add your own brand logo during a live broadcast, you can add a logo image. As shown in the image below, this is the effect after adding a transparent background logo image. The newly added image will have a yellow border around it, indicating that it is currently selected. A selected multimedia source can be moved and resized with mouse. It can also be rotated and modify its display level by right-click menu.



3. **Start a live broadcast**

Click '**Go Live**' button to start a live broadcast . Once the broadcast starts successfully, the **Go Live** button will turn into **End**. Click it to end the live boradcast.

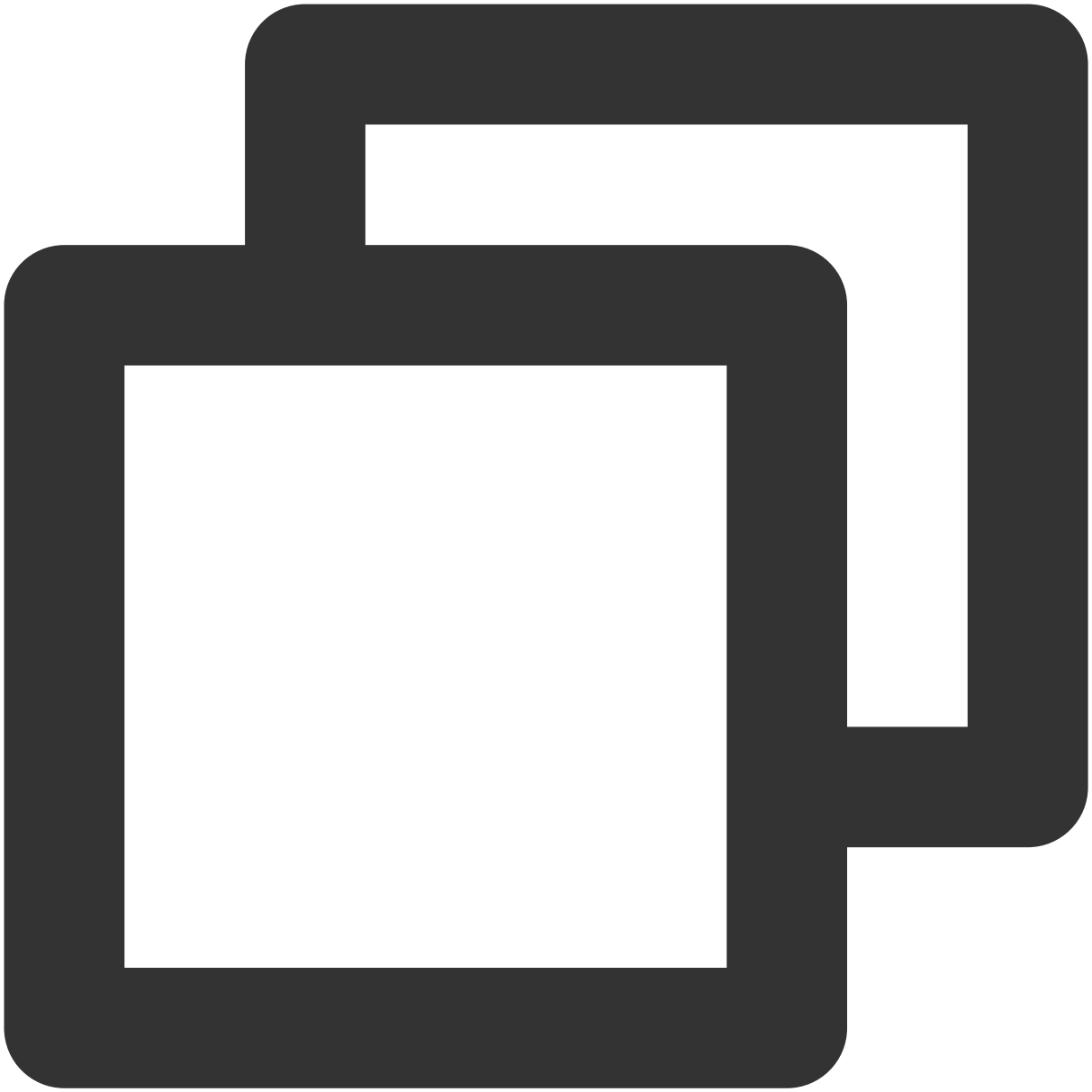


4. **View the live broadcast**

The desktop version only supports the host starting the broadcast. To watch, you need to use the mobile app. Find the corresponding live room in the live list on the mobile app and enter the live room. For the use of the mobile app, please refer to the documentation foriOSand Android.

Step 6: Build installation package

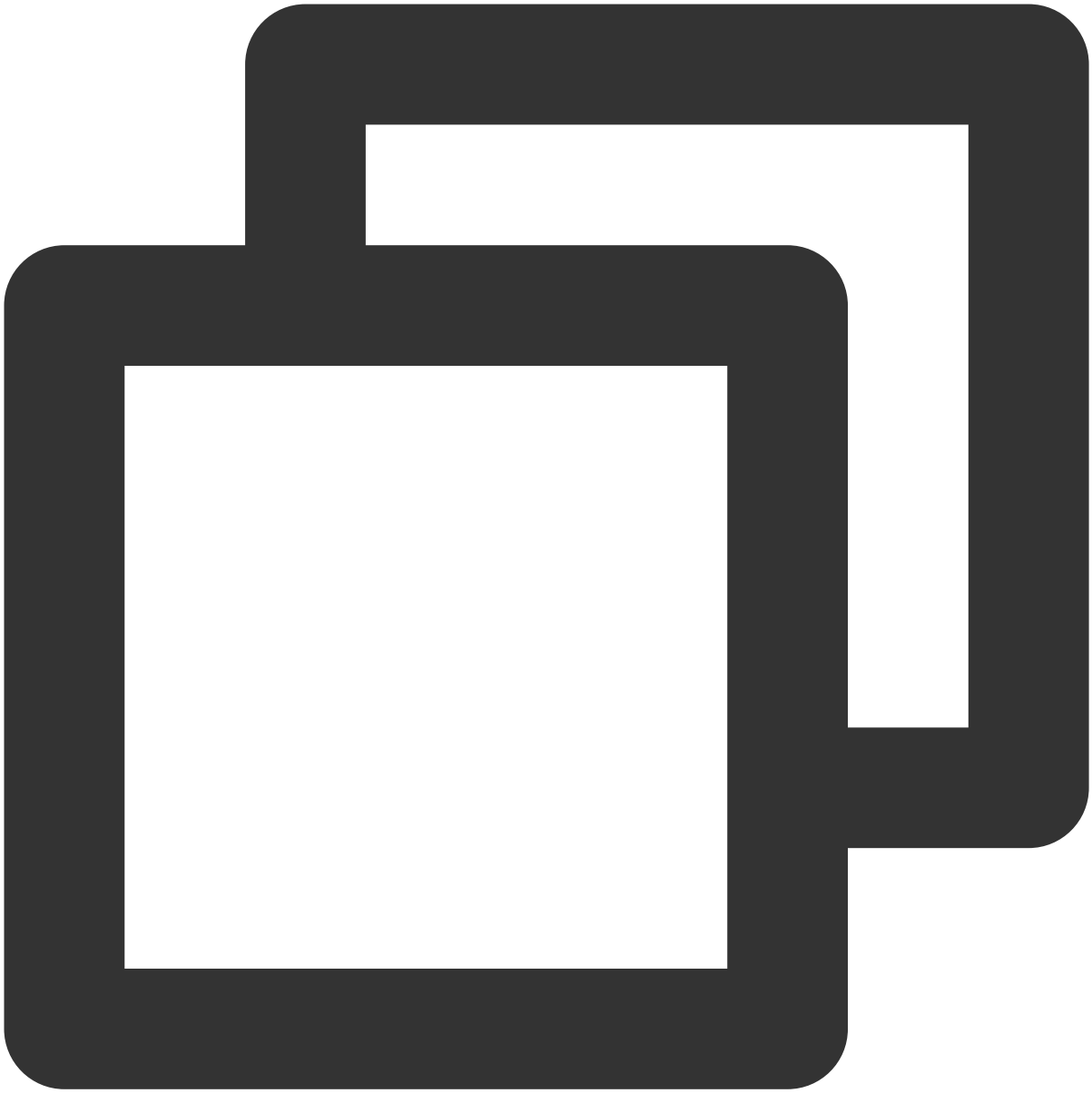
1. Copy `ultra-live-electron/electron-builder.json5` file into your project's root directory. You can modify the `productName` and `appId` as you like.
2. Add the following command in your project's package.json file to enable buliding installation package.



```
{
  "scripts": {
    "build:win64": "electron-builder --win --x64",
    "pack:win64": "npm run build && npm run build:win64"
  }
}
```

3. Build installation package

Enter the root path of your project with `cmd.exe` and execute the following command. The created installation package is in `release` directory.



```
npm run pack:win64
```

4. Install the pacakge and run.

Common questions

Have no project meets the integration requirement, How can I start?

There are several ways to start your TUILiveKit journey:

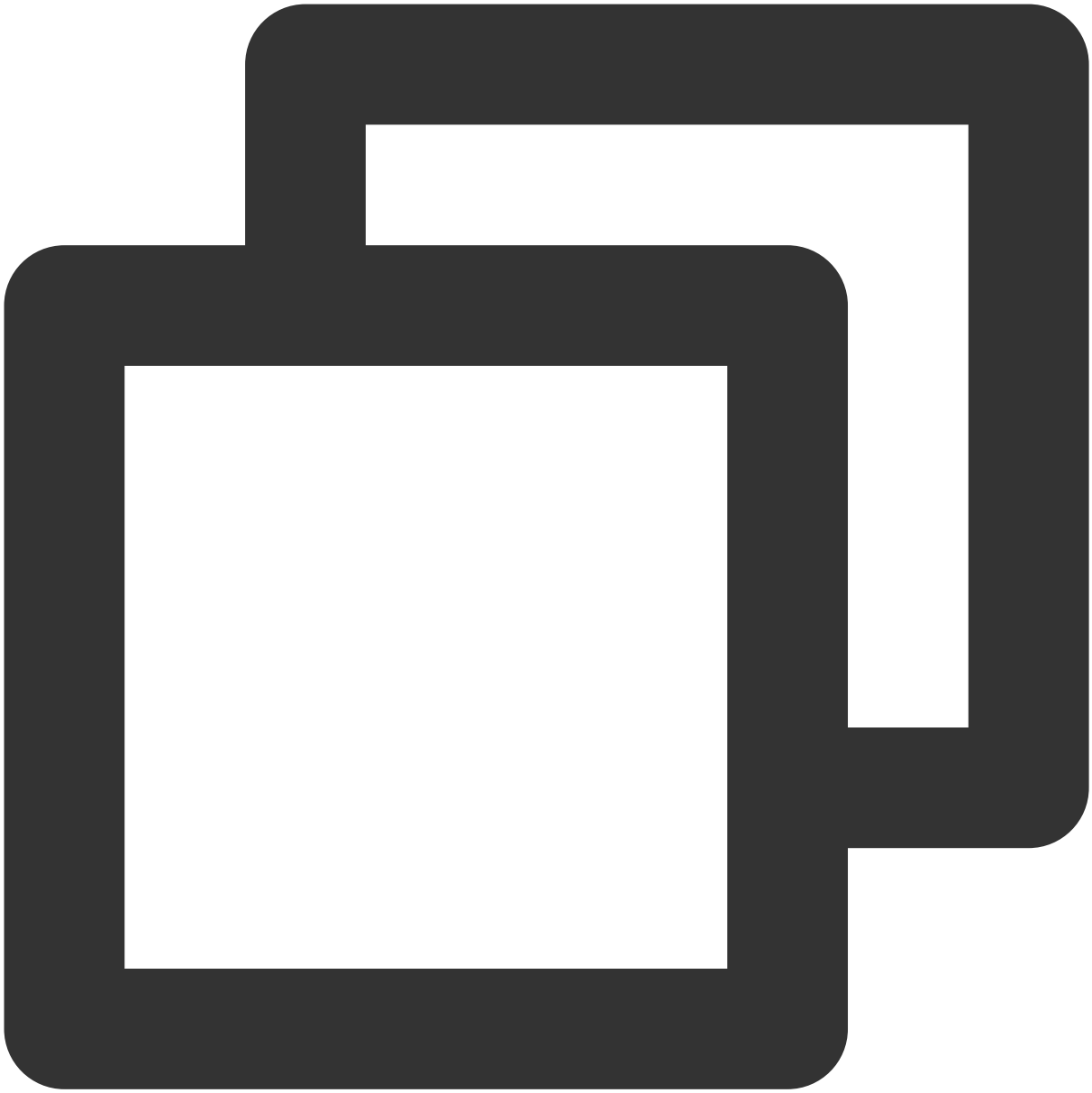
If you do not have a project, then you can just start with our open source Github . Just clone it and modify the code as you need.
If you already have a project, you can adjust your project as our open source Template project: trtc-electron-template-vue3-webpack to ingerated TUILiveKit.
If you have a JavaScript project withour TypeScript support, you can refer to "How to integrate TUILiveKit in JavaScript project?".

Does TUILiveKit support Vite?

Currently, TUILiveKit does not support running with Vite.

How to integrate TUILiveKit in JavaScript project?

JavaScript project can not integrate TUILiveKit directly. It should be modify to support TypeScript to integrate TUILiveKit.
1. Install dependencies



```
npm install --save-dev typescript@4.5.5 @typescript-eslint/eslint-plugin@5.4.0 @typ
```

2. Copy `ultra-live-electron/tsconfig.json` file into your project root diectory.

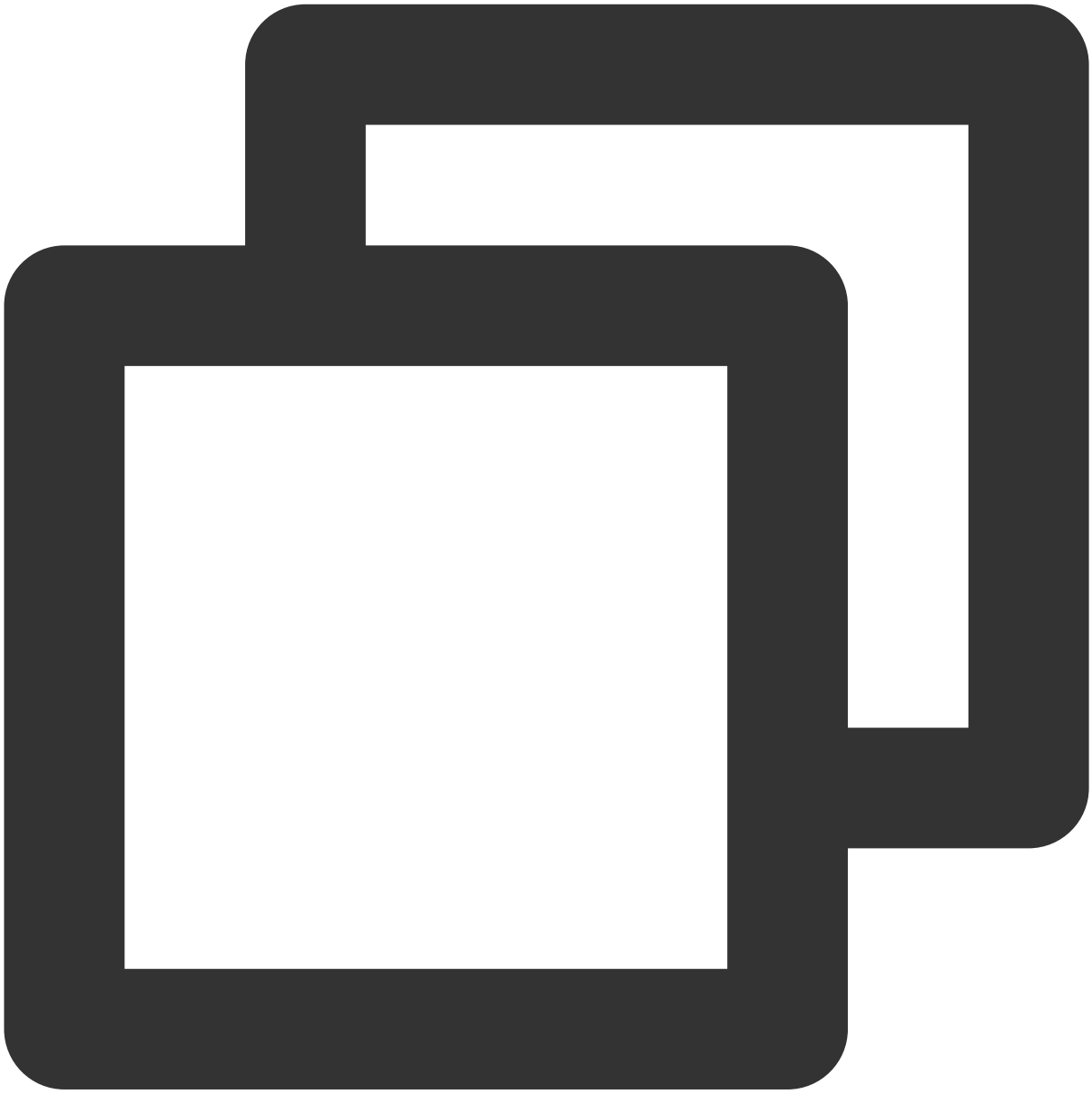
3. Copy `ultra-live-electron/src/global.d.ts` file into your project root diectory.

How to Enable the Beauty Function?

The humain beauty capability in TUILiveKit is base on Tencent Effect SDK, which you have to buy and activate to abtain the `licenseURL` and `licenseKey` to use. Enter the `licenseURL` and `licenseKey` into `src/TUILiveKit/utils/beauty.ts` file to quick start.

Note：

For production projects, it is necessary to obtain the `licenseURL` and `licenseKey` by calling the backend service. Writing them into a JavaScript file can enable quick startting, but there is a very high risk of leakage for both `licenseURL` and `licenseKey` . This method is only suitable for quick integration and testing purposes.



```
// beauty.ts
export const XmagicLicense = {
  licenseURL: "",
  licenseKey: "",
};
```

Suggestions and Feedback

If you have any suggestions or feedback, please contact info_rtc@tencent.com.

Flutter

Last updated：2024-08-13 17:40:41

Environment Preparations

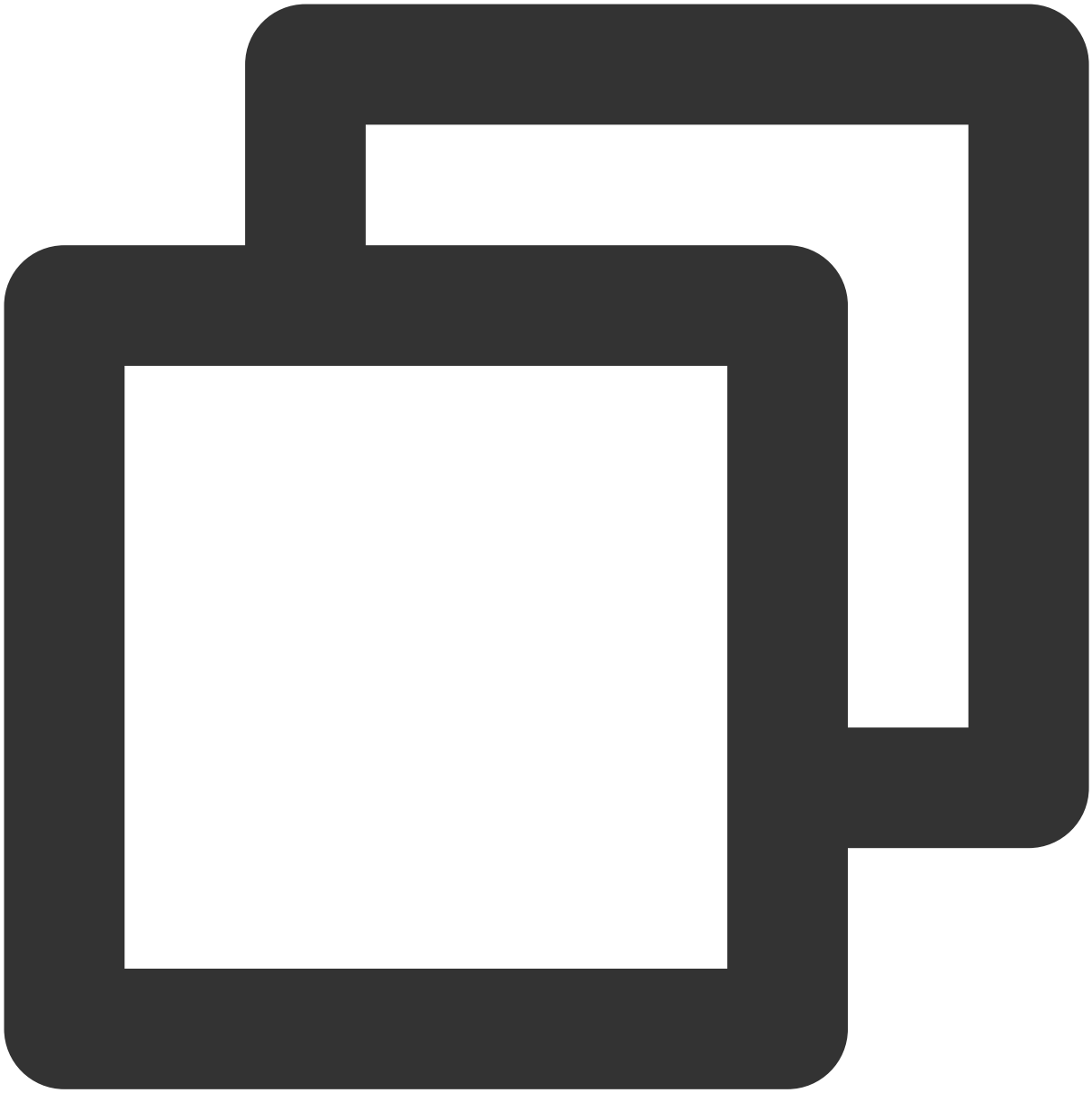
Platform	Version
Flutter	Flutter 3.22.0 or later.。 Dart version 3.4.0 or higher.
Android	Android Studio 3.5 or later. Android devices 5.0 or later.
iOS	Xcode 13.0 or later. Please ensure that your project has a valid developer signature set.

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. Import the TUILiveKit component

From the root directory of the project, install the component tencent_live_uikit plug-in by executing the following command from the command line.



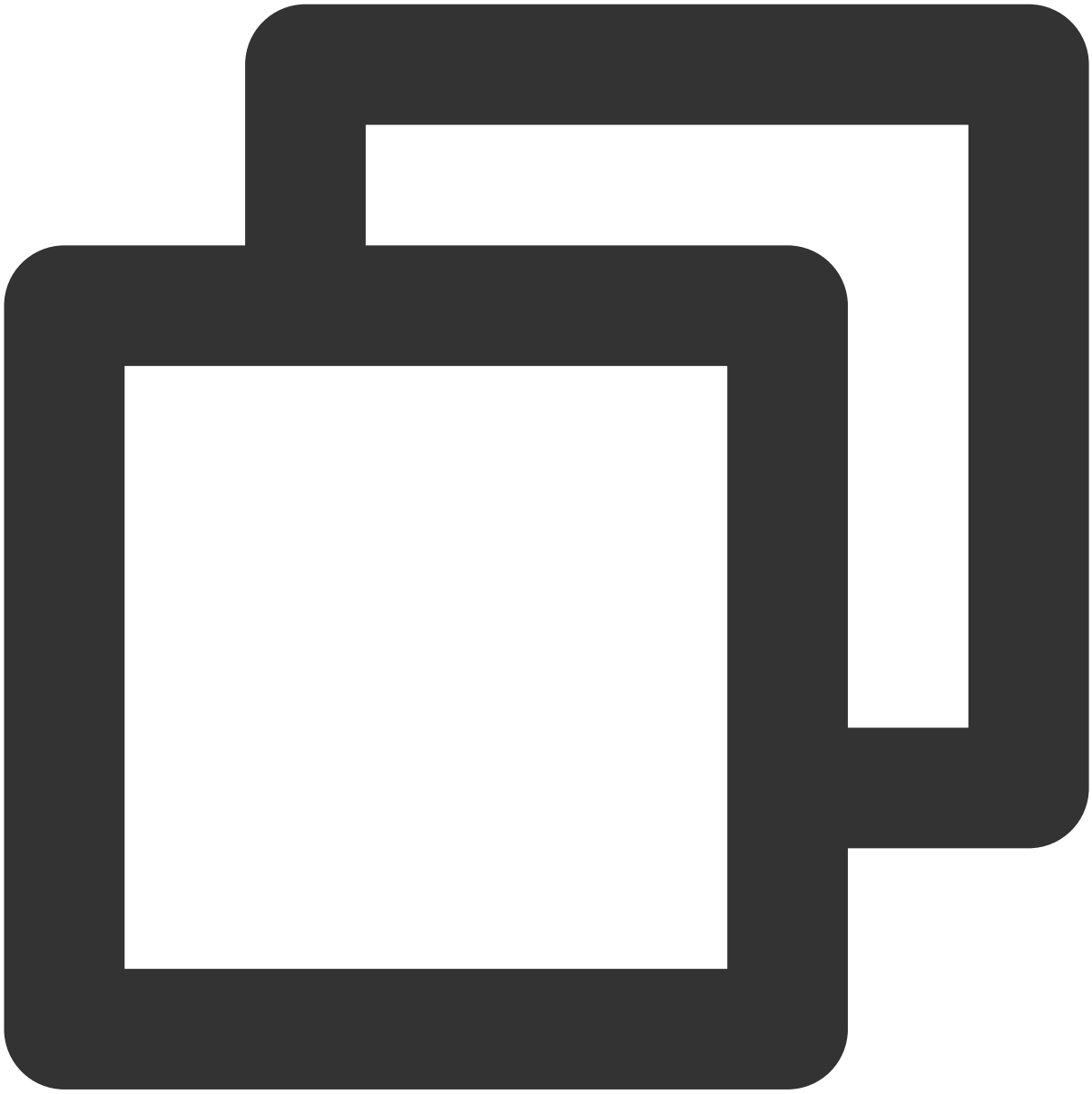
```
flutter pub add tencent_live_uikit
```

Step 3. Complete the project configuration

Android

iOS

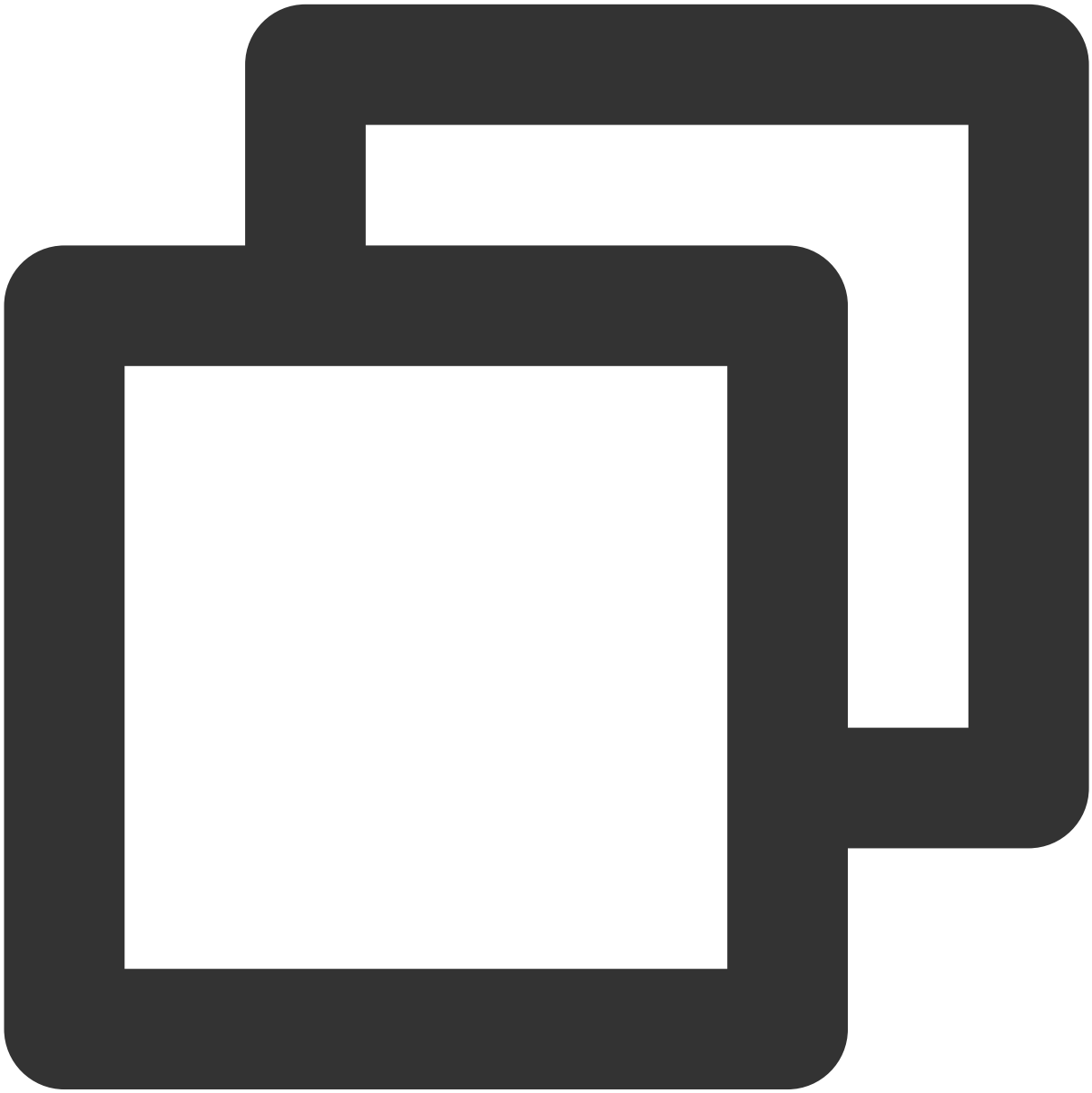
1. If you need to compile and run on the Android platform, because we use Java's reflection features inside the SDK, you need to add some classes in the SDK to the non-confusion list.
First, you need to configure and enable the obfuscation rule in your project's `android/app/build.gradle` file:



```
android {
  ......
  buildTypes {
      release {
          ......
          minifyEnabled true
          proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard
```

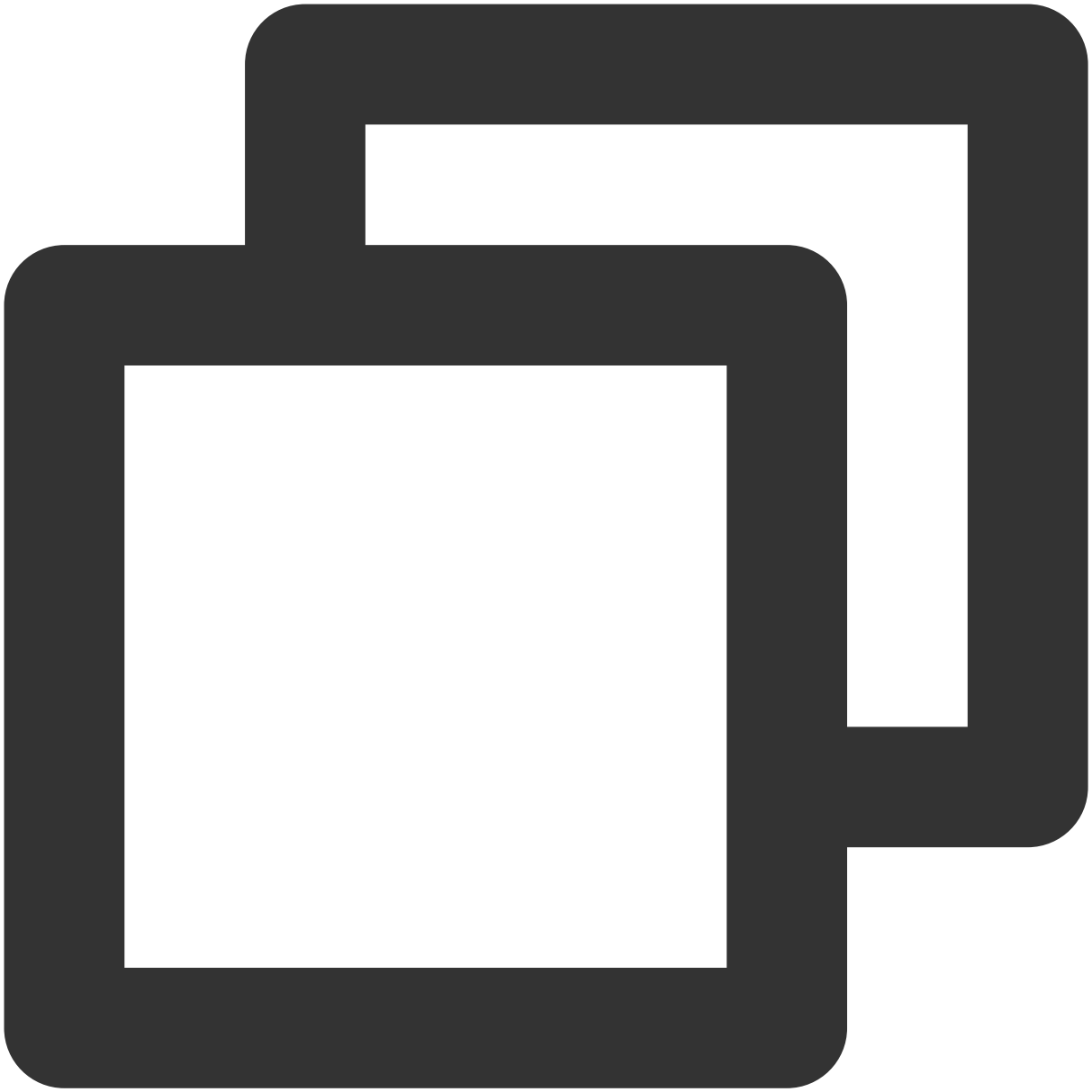
```
        }
    }
}
```

Create a `proguard-rules.pro` file in the `android/app` directory of the project, and add the following code in the `proguard-rules.pro` file:



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

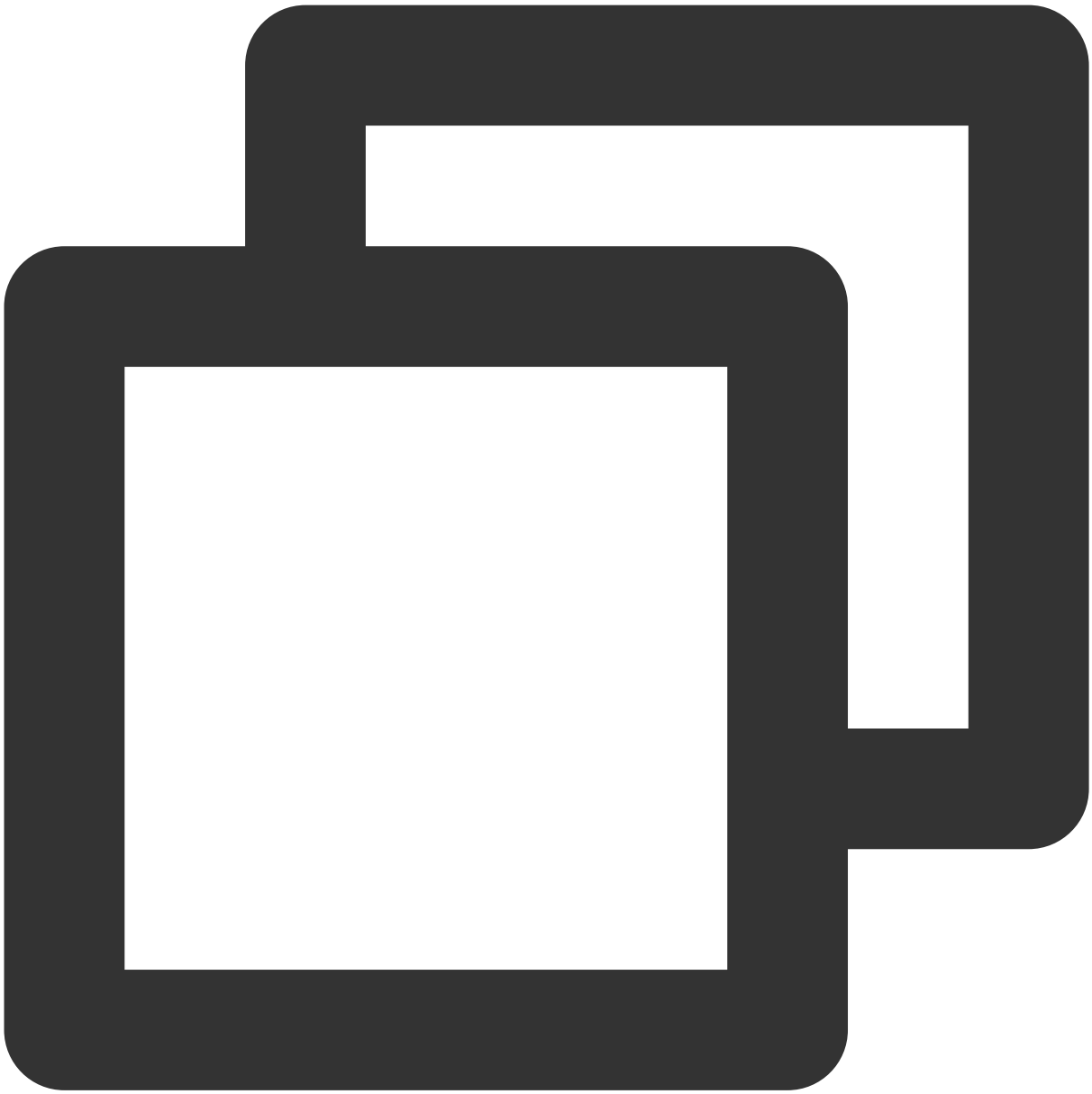
2. Configure to enable Multidex support in the `android/app/build.gradle` file of your project



```
  android {
    ......
    defaultConfig {
      ......
      multiDexEnabled true
    }
}
```

1. Use **Xcode** to open your project, select **Item** > **Building Settings** > **Deployment**, and set the **Strip Style** to `Non-Global Symbols` to retain the necessary global symbol information.

2. **Optional** If you need to debug on the iOS Emulator and you are using a Mac computer with an Intel Chip, you need to add the following code in the project's `ios/Podfile` file:

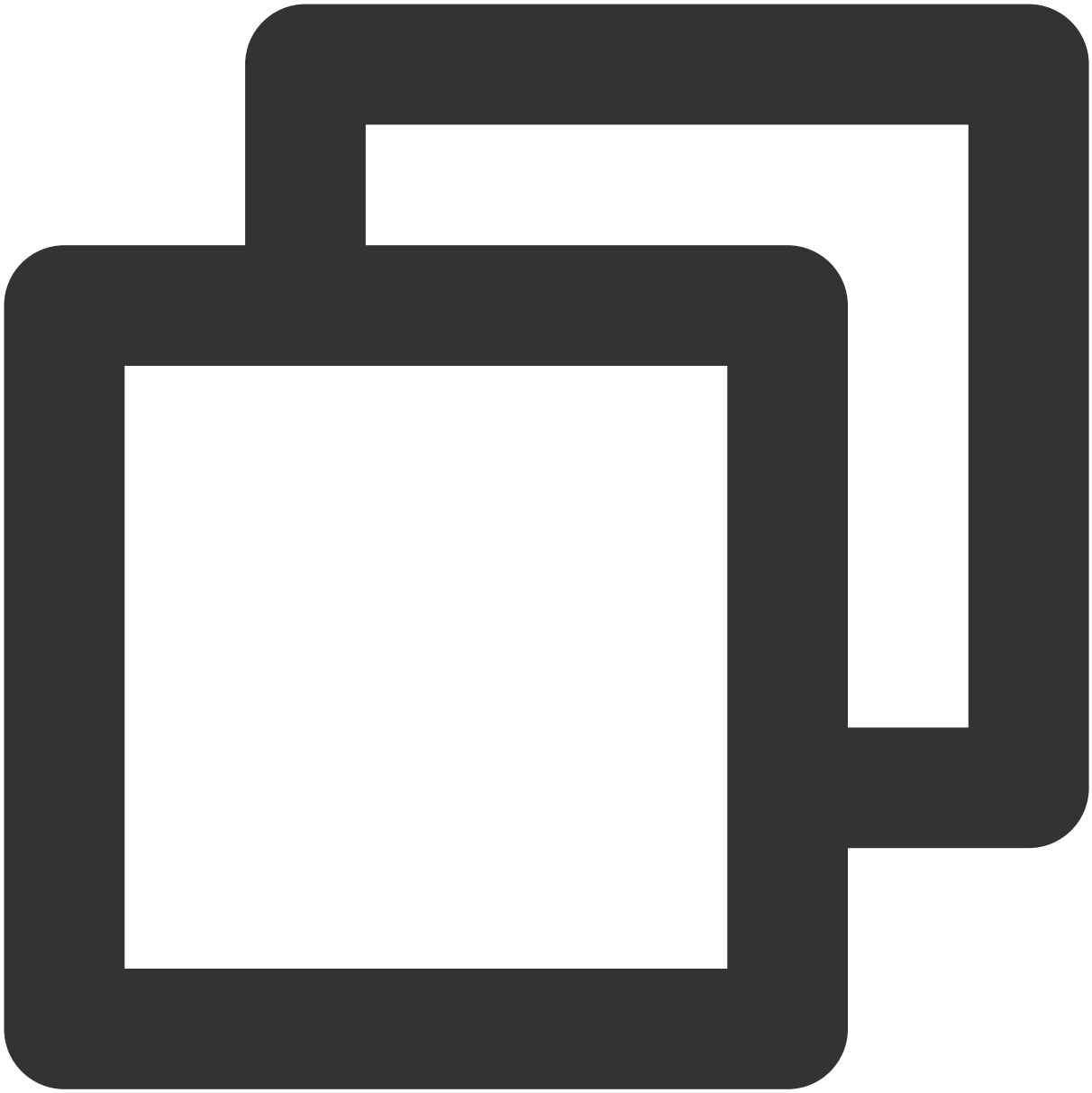


```
target 'xxxx' do
  ......
end
......

post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
```

```
    target.build_configurations.each do |config|
      config.build_settings['VALID_ARCHS'] = 'arm64 arm64e x86_64'
      config.build_settings['VALID_ARCHS[sdk=iphonesimulator*]'] = 'x86_64'
    end
  end
end
```

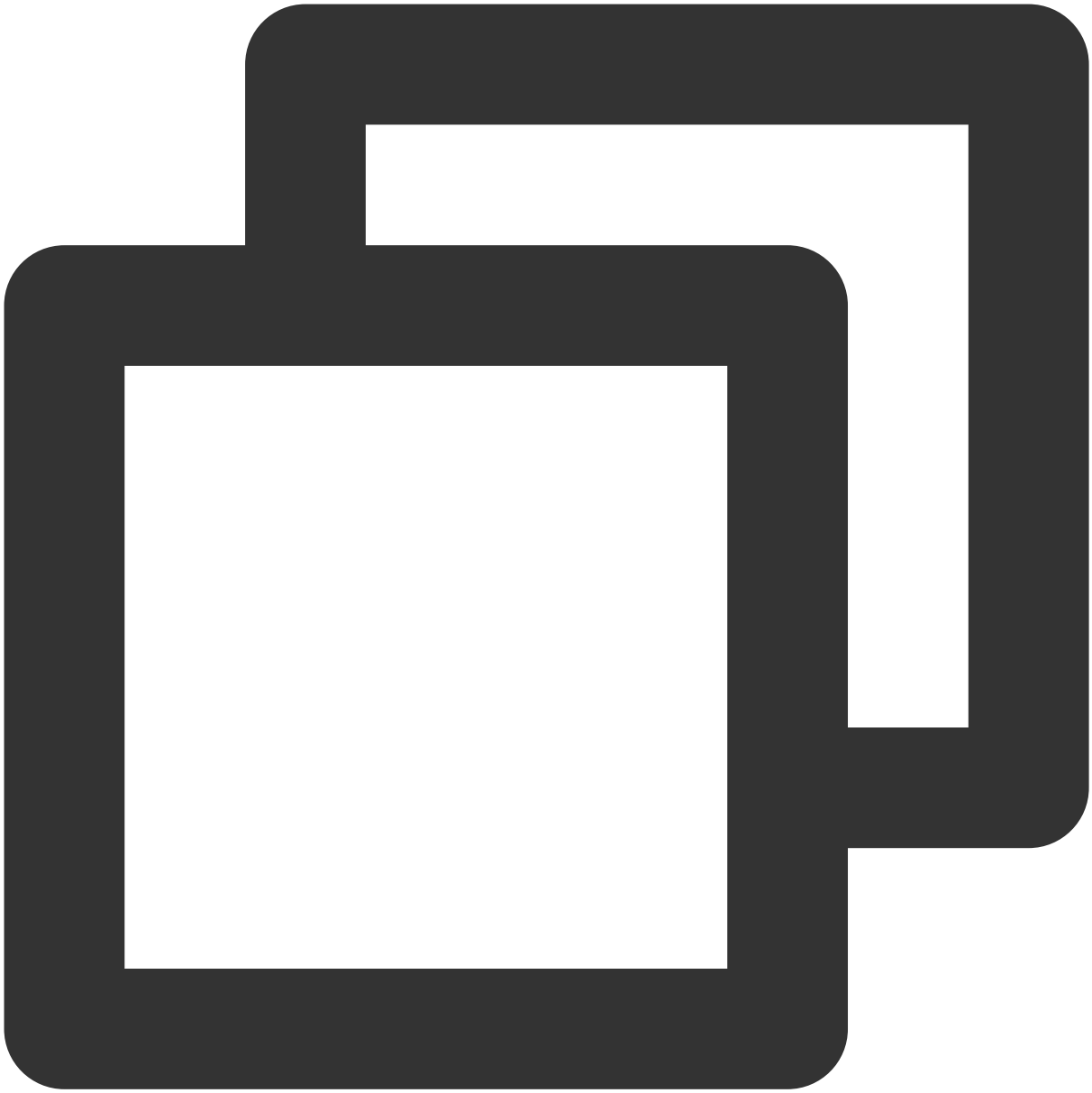
3. Since TUILiveKit will use iOS's audio and video features, you need to grant permissions for the microphone and camera.

Authorization Operation Method: In your iOS project's `Info.plist` , under the first-level `<dict>` directory, add the following two items. They correspond to the system's prompt messages when asking for microphone and camera permissions.



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

After completing the above additions, add the following preprocessor Definition in your **ios/Podfile** , to enable camera and microphone permissions.

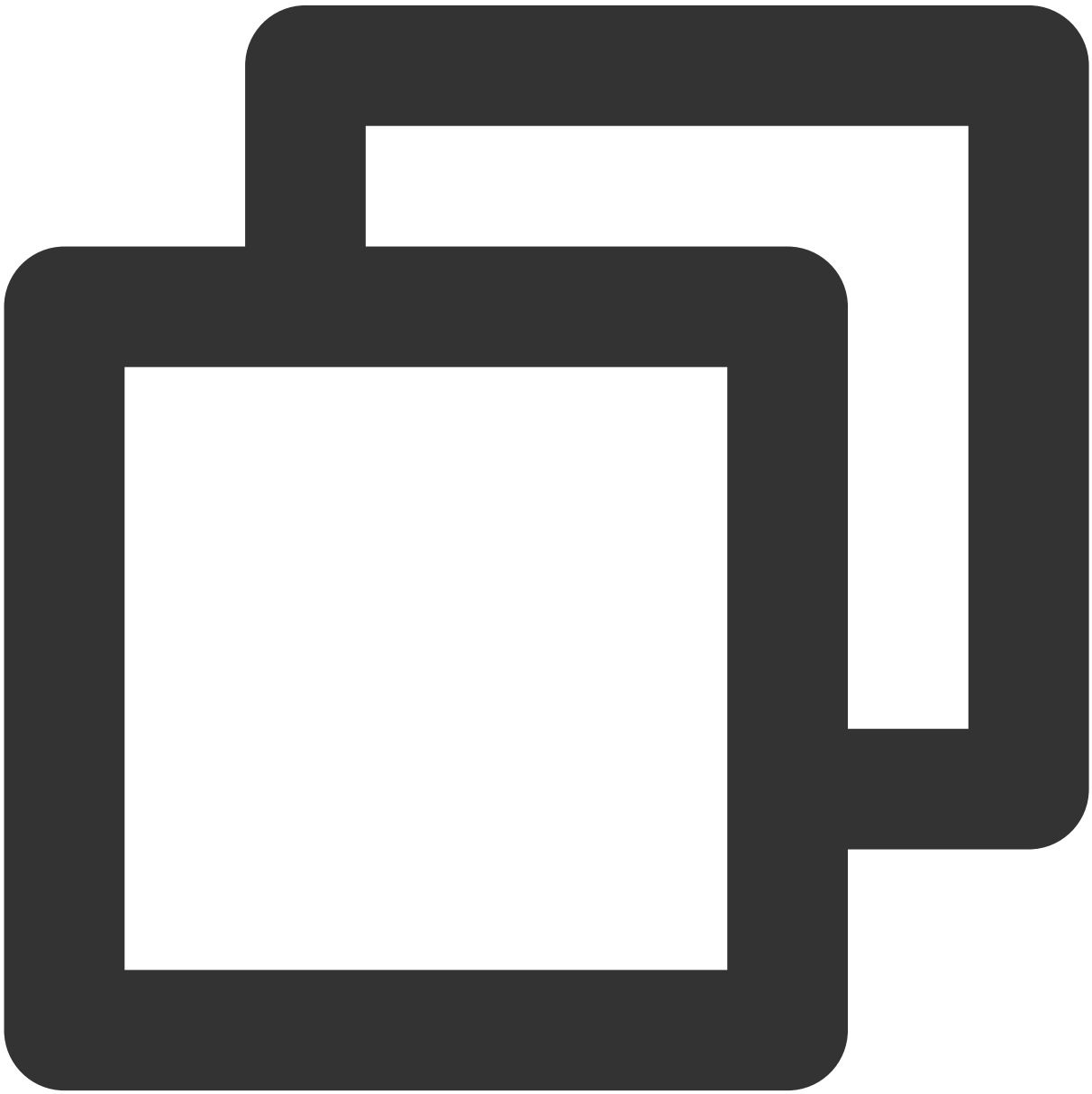


```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
      target.build_configurations.each do |config|
        config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
      '$(inherited)',
      'PERMISSION_MICROPHONE=1',
      'PERMISSION_CAMERA=1',
      ]
    end
  end
```

```
end
```

Step 4. Set up navigatorObserver and localizationsDelegates

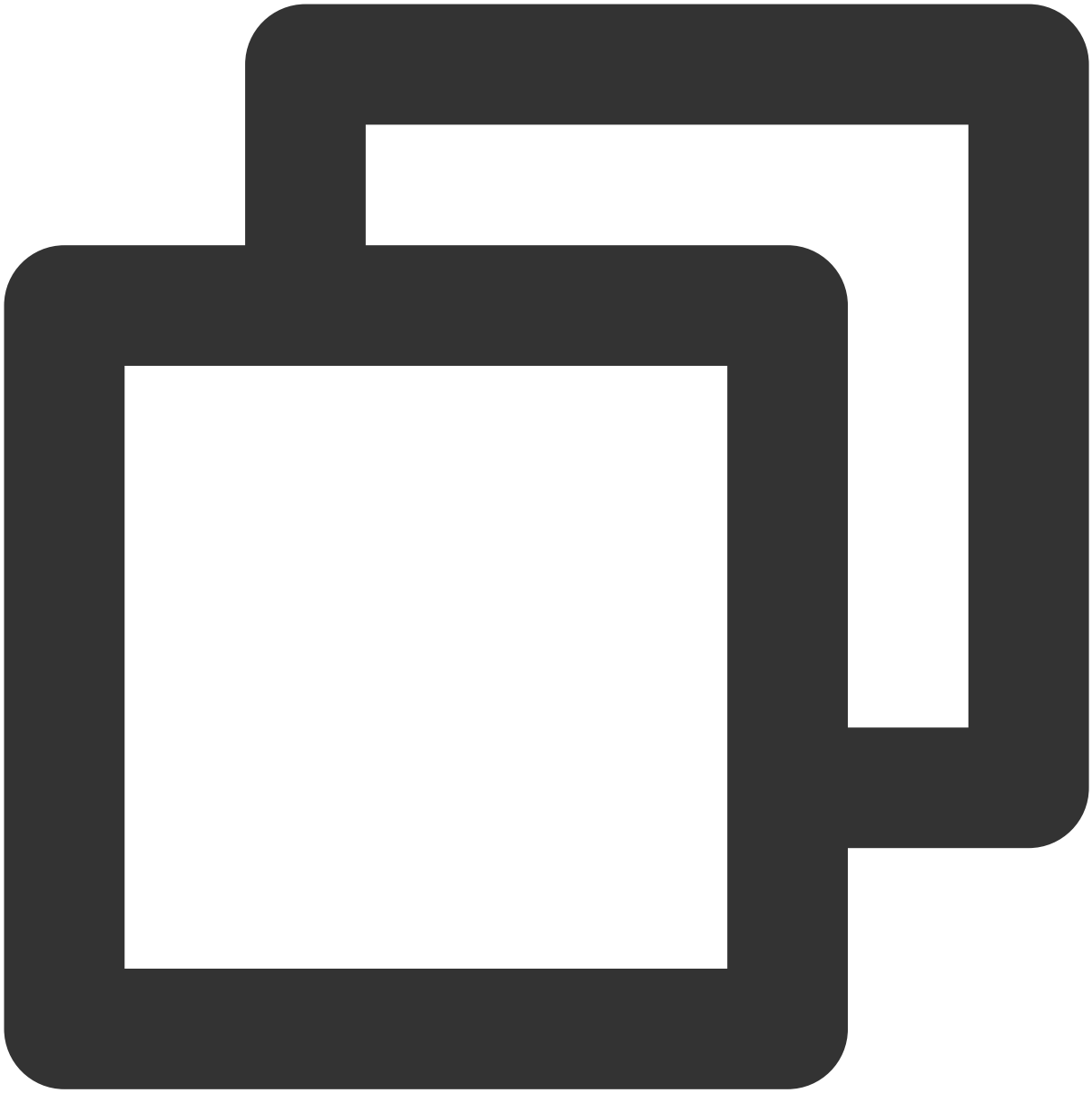
In the Flutter application framework, add TUICallKit.navigatorObserver to navigatorObservers, and add LiveKitLocalizations.localizationsDelegates to localizationsDelegates. For example, using the MaterialApp framework, the code is as follows:



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

 ......

class XXX extends StatelessWidget {
  const XXX({super.key});

 @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorObservers: [TUILiveKitNavigatorObserver.instance],
      localizationsDelegates: [...LiveKitLocalizations.localizationsDelegates],
      ......
    );
  }
}
```

Step 5. log in to TUILiveKit componet

Before using the various features of the TUILiveKit component, you must first execute the TUI component's log in. In your project, it is recommended to add the following log in code in your business log in scenario or the first launch Activity of the App. Its function is to complete the log in of the TUI component by calling the relevant interfaces in TUICore. This step is extremely critical, as you can only use the various features of TUILiveKit normally after successfully logging in. Therefore, please patiently check whether the relevant parameters are configured correctly:



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

login() async {
  await TUILogin.instance.login(
    1400000001,     // Please replace with the SDKAppID obtained from step one
    "denny",        // Please replace with your UserID
    "xxxxxxxxxxx",  // You can calculate a UserSig in the console and fill it in he
    TUICallback(
      onError: (code, message) {
        print("TUILogin login fail, {code:$code, message:$message}");
```

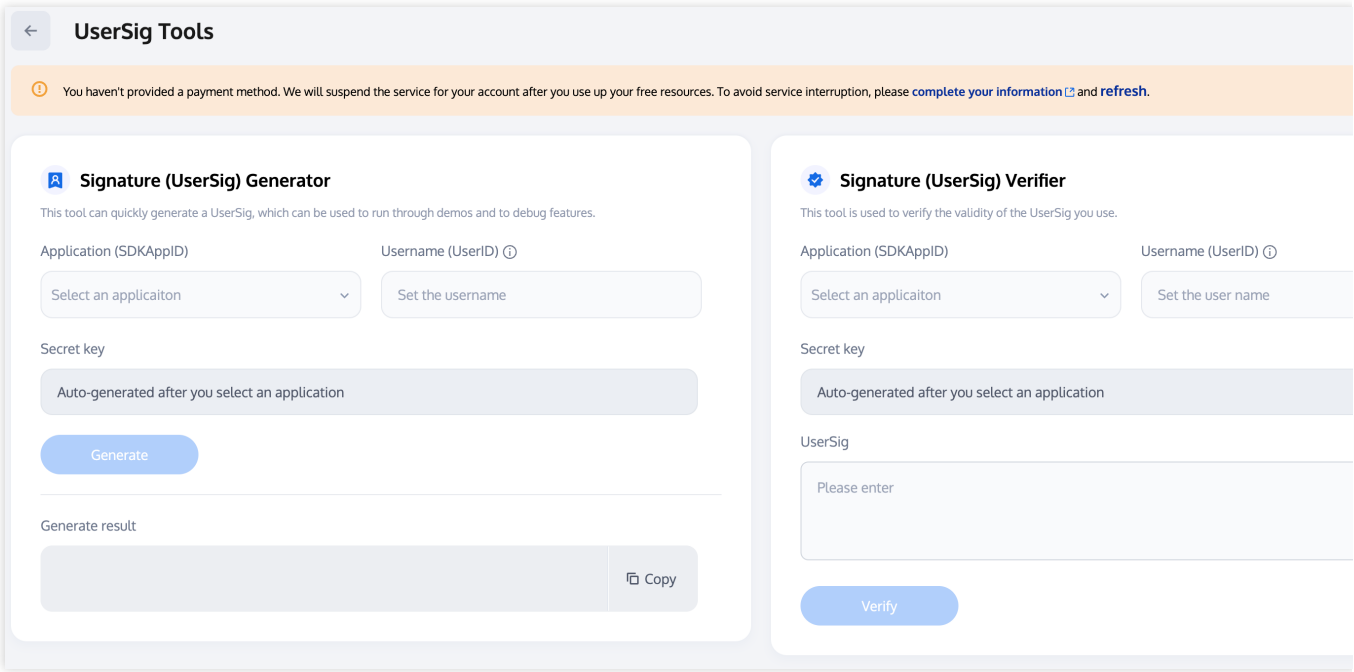
```
        },
        onSuccess: () async {
          print("TUILogin login success");
        },
      ),
    );
  }
```

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.

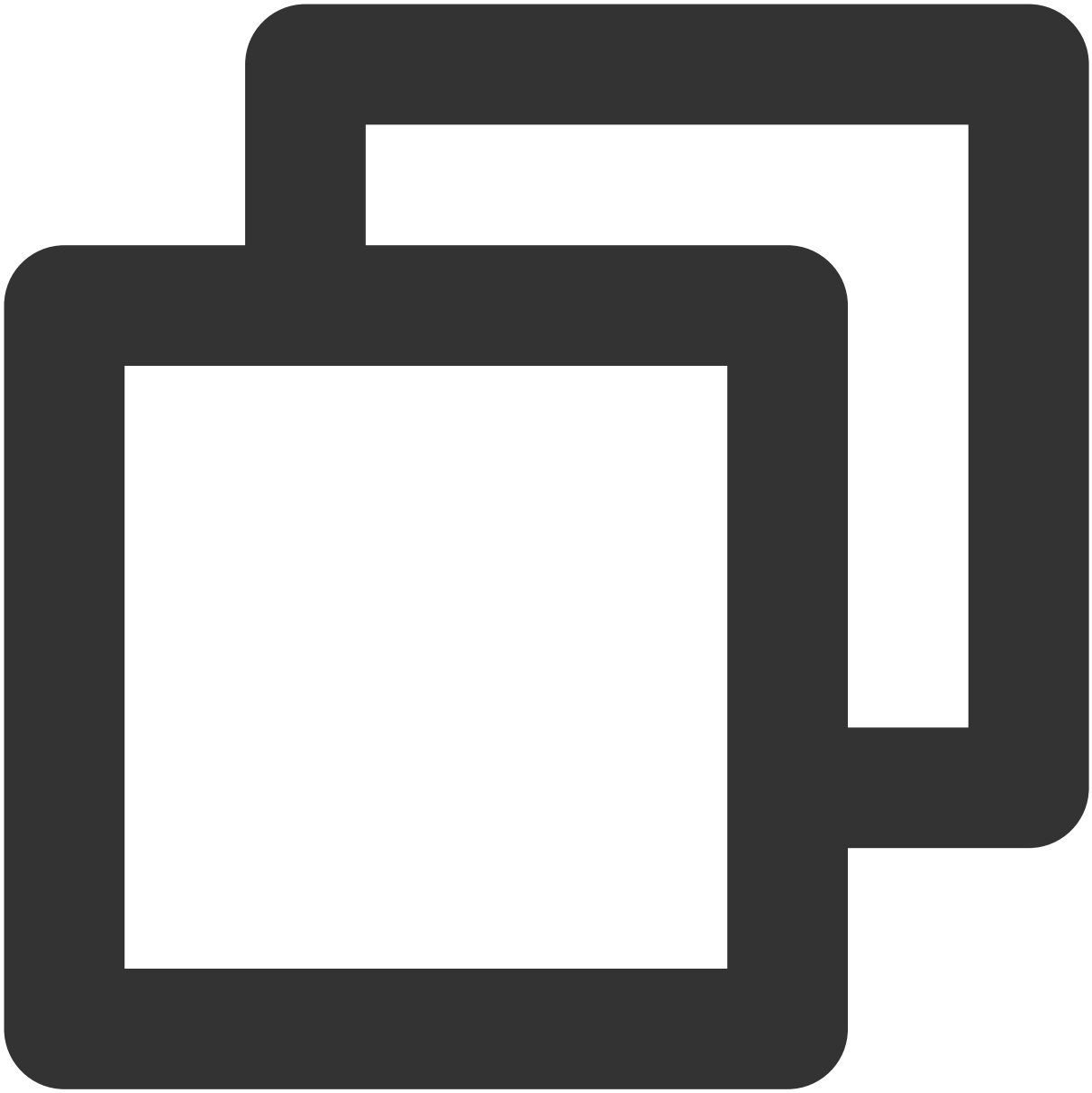
userSig is set to the value of Secretkey mistakenly. The userSig is generated by using the SecretKey for the purpose of encrypting information such as sdkAppId, userId, and the expiration time. But the value of the userSig that is required cannot be directly substituted with the value of the SecretKey.

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 6. Enter the live preview screen

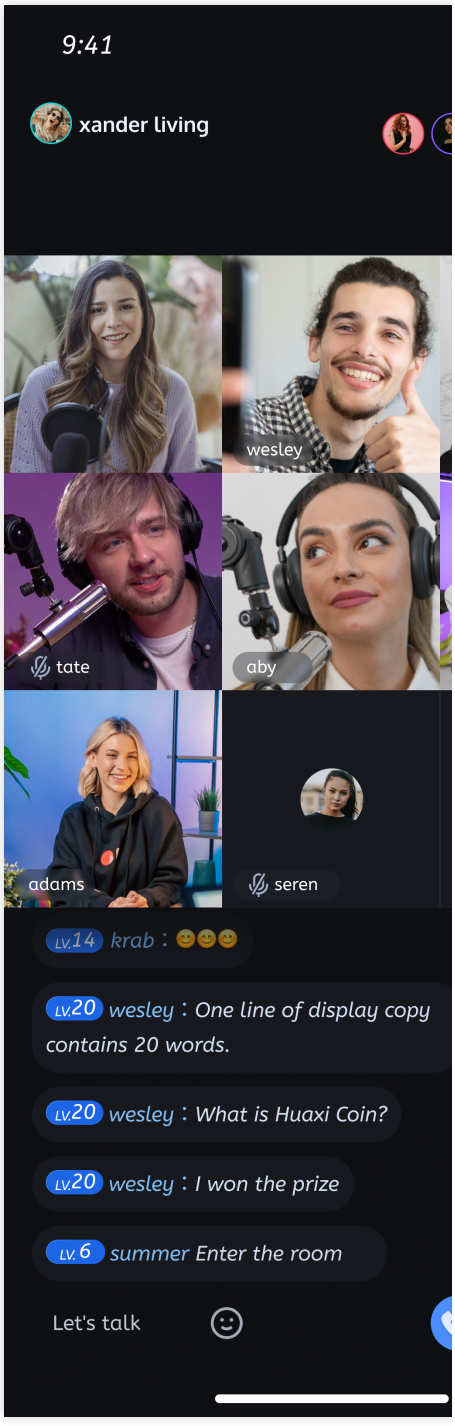
Note:

Please make sure to follow Step 5 and complete the log in to actio
Where you need to start the live streaming (as determined by your business, execute it within its click event), perform the following operations to launch the broadcaster's live streaming page:



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

Navigator.push(context, MaterialPageRoute(
  builder: (context) {
    return TUILiveRoomAnchorWidget(
        roomId: LiveIdentityGenerator.instance.generateId(AppStore.userId, RoomType
  },
));
```

Video Live Preview Screen



Live video streaming with pictures

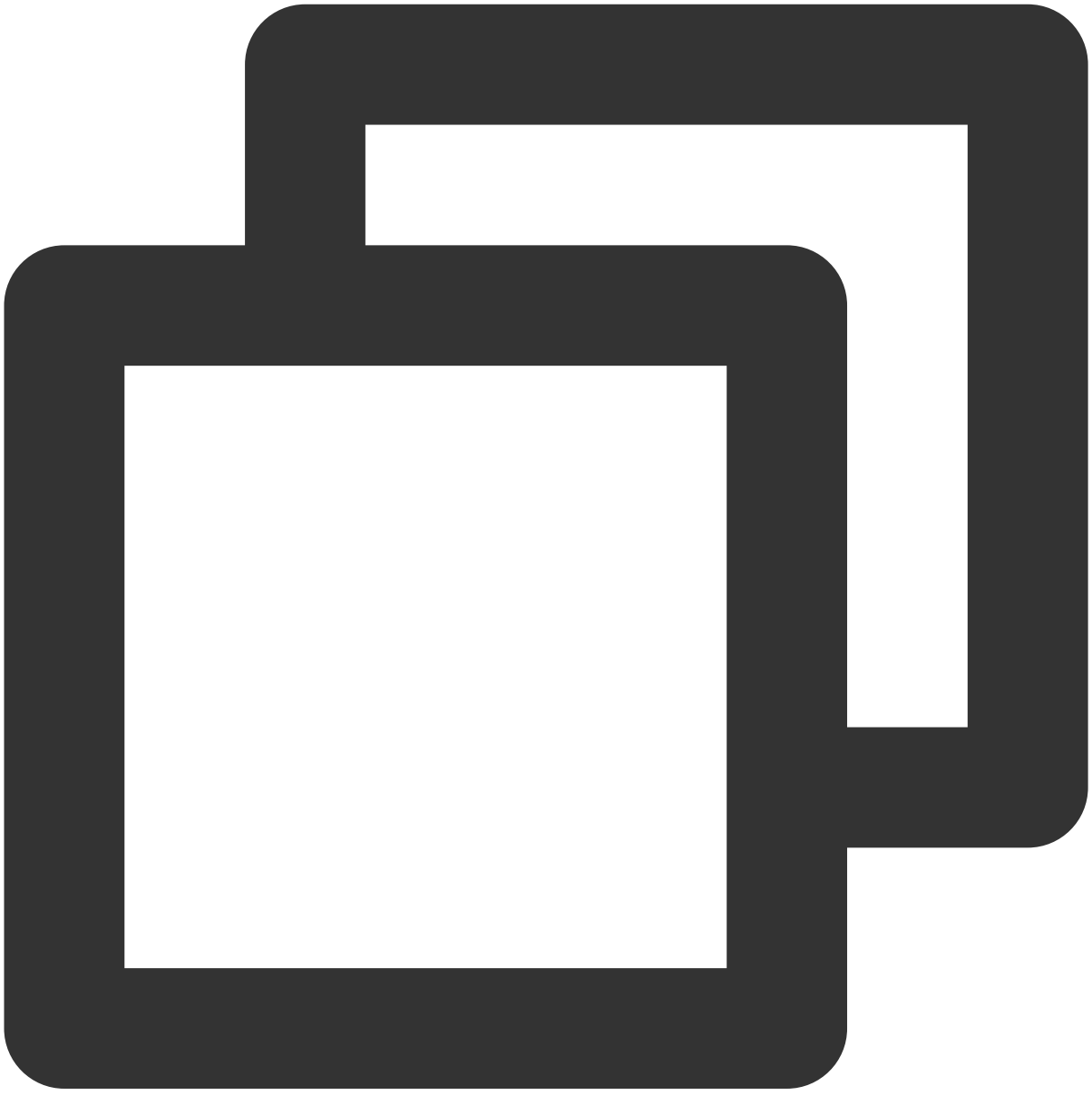
Step 7. Pull the room list

Note:

Please make sure to follow Step 5 and complete the log in to action. Only after `TUILogin.login` log in to is successful, you can enter the live preview screen no

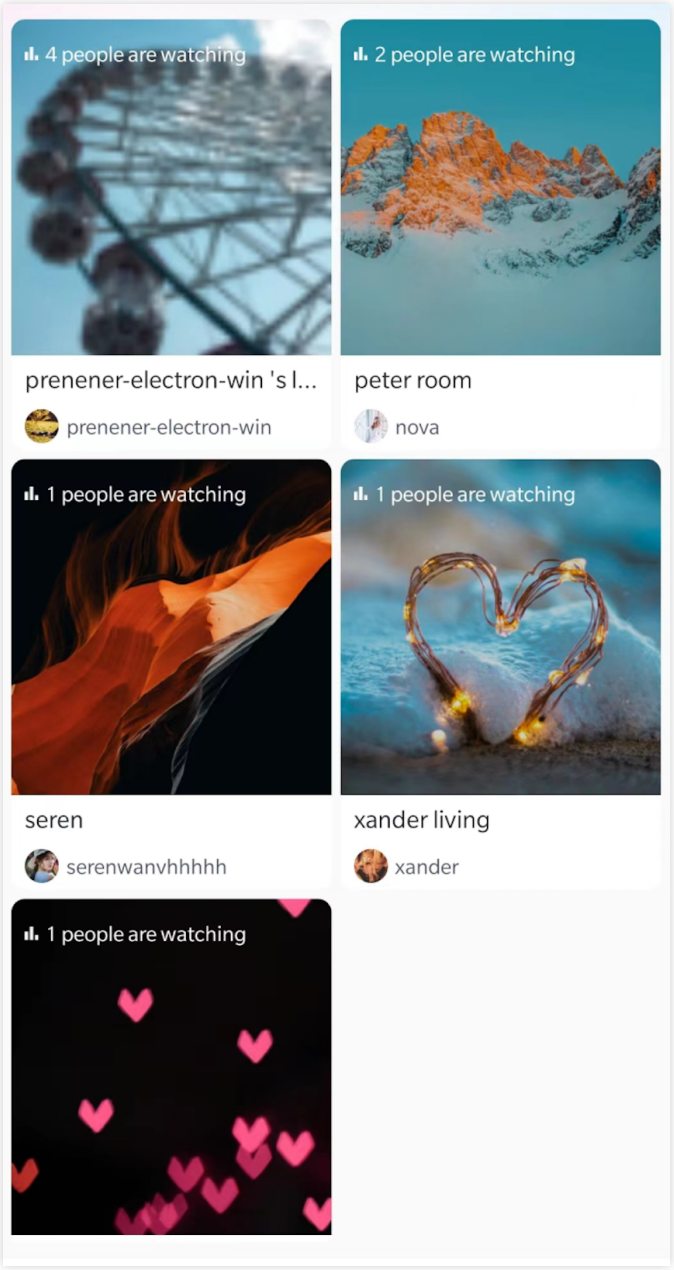
In your widget, by loading the TUILiveKit's LiveListWidget component, you can display the room list.

Java



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


return Scaffold(
  body: SizedBox(
```

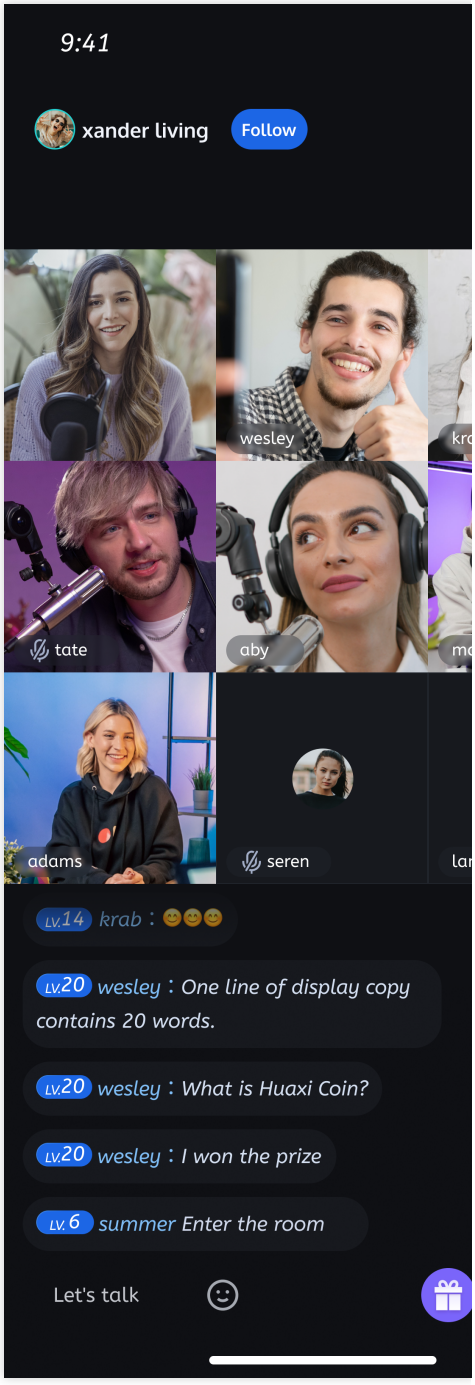
```
      width: _screenWidth,
      height: double.infinity,
      child: LiveListWidget(),     // Adding the room list component LiveListWidget of
    ),
  );
```



Step 8. Enter the room as audience

In the Step 7 room list interface, click any room to automatically enter the live streaming room.

Video Live Room



Video Live Room

Suggestions and Feedback

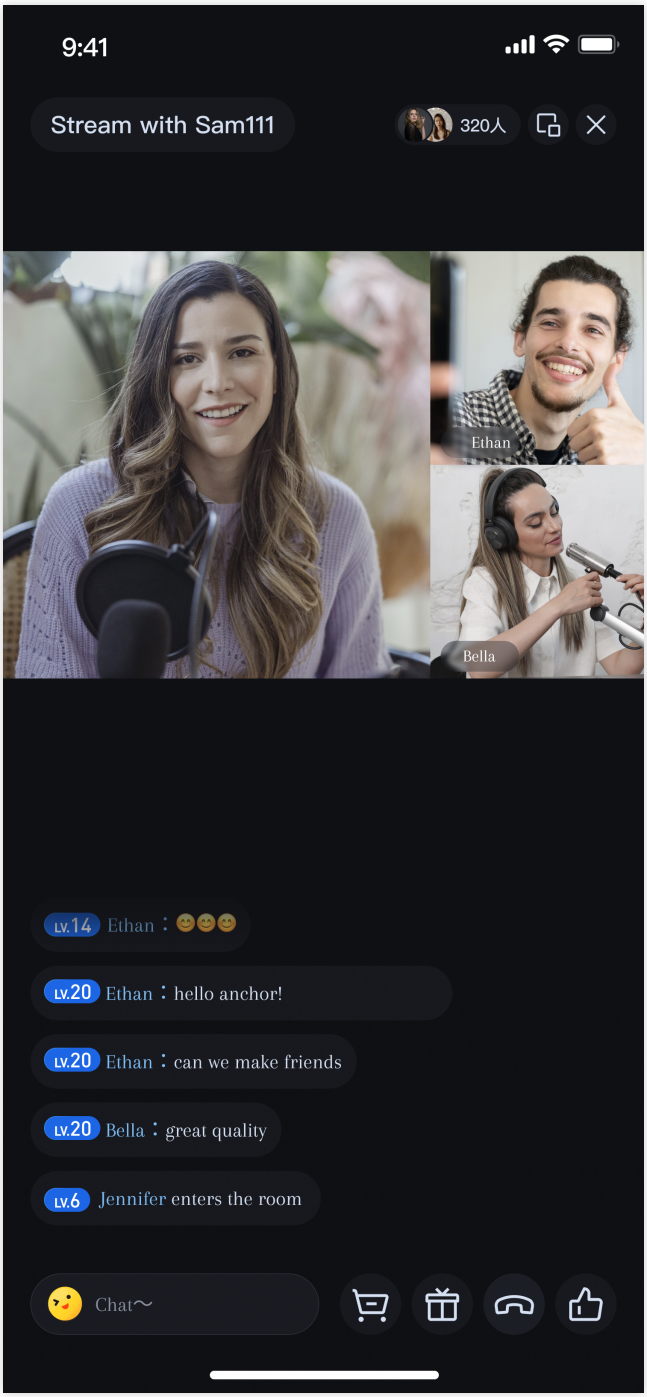
If you have any suggestions or feedback, please contact info_rtc@tencent.com.

Interactive Bullet Comments (TUILiveKit) iOS

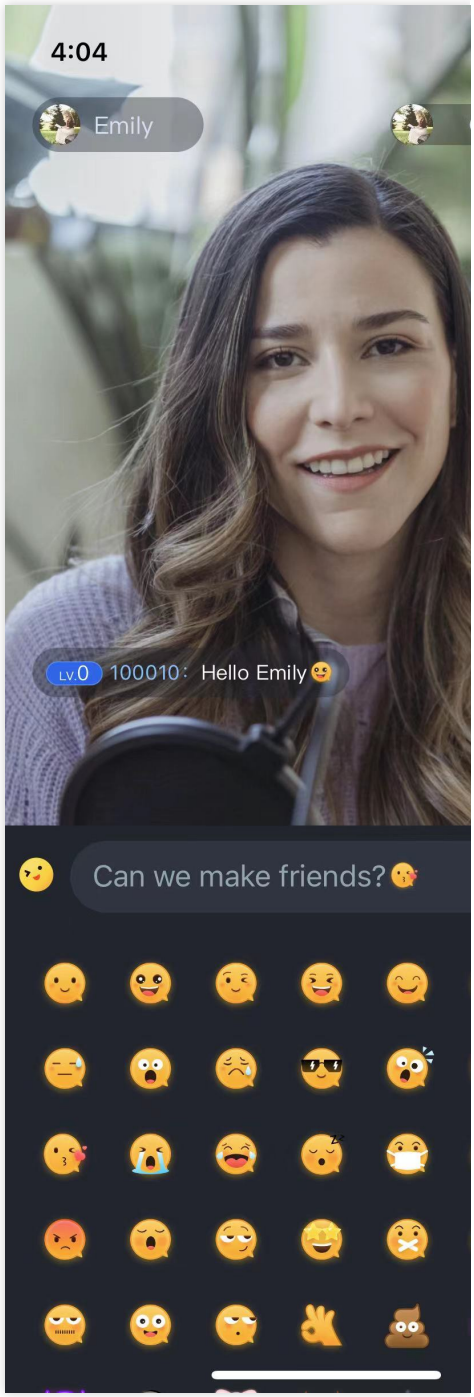
Last updated：2024-08-09 22:25:01

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**.

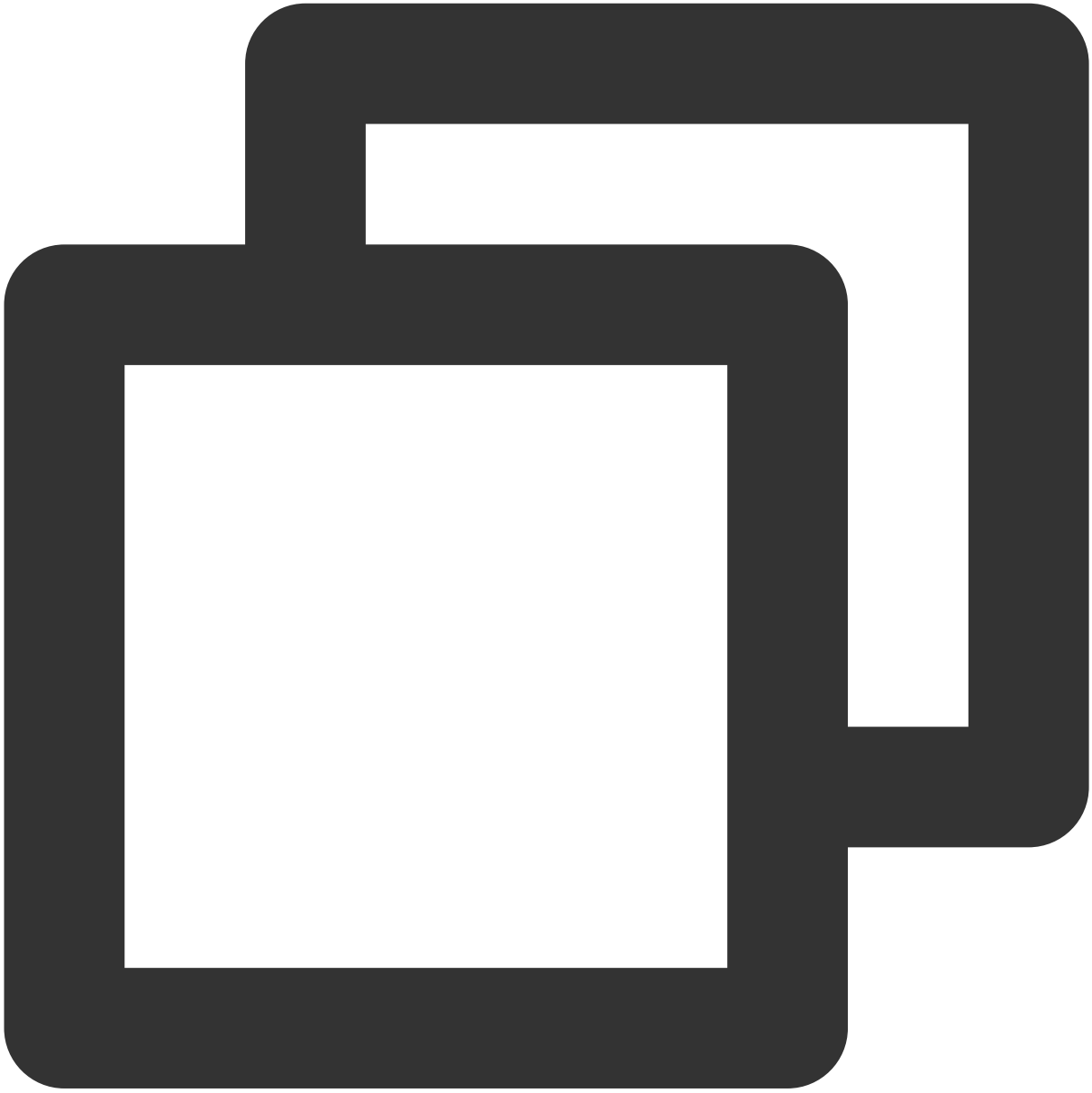
Integration

The barrage component mainly provides two Objects：

`TUIBarrageButton` : Clicking it can bring up the input interface.

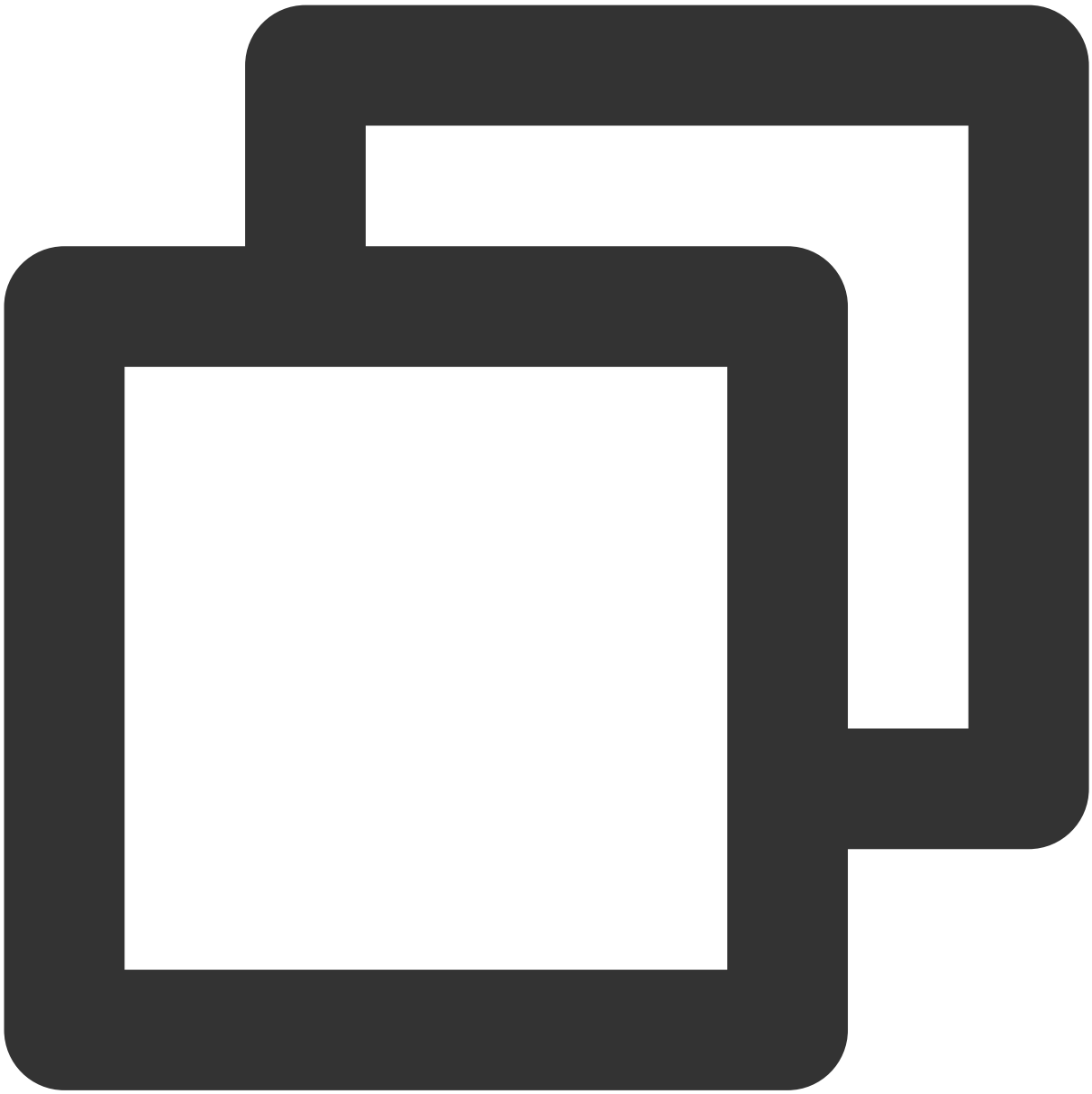
`TUIBarrageDisplayView` : Used for displaying barrage messages.

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
```

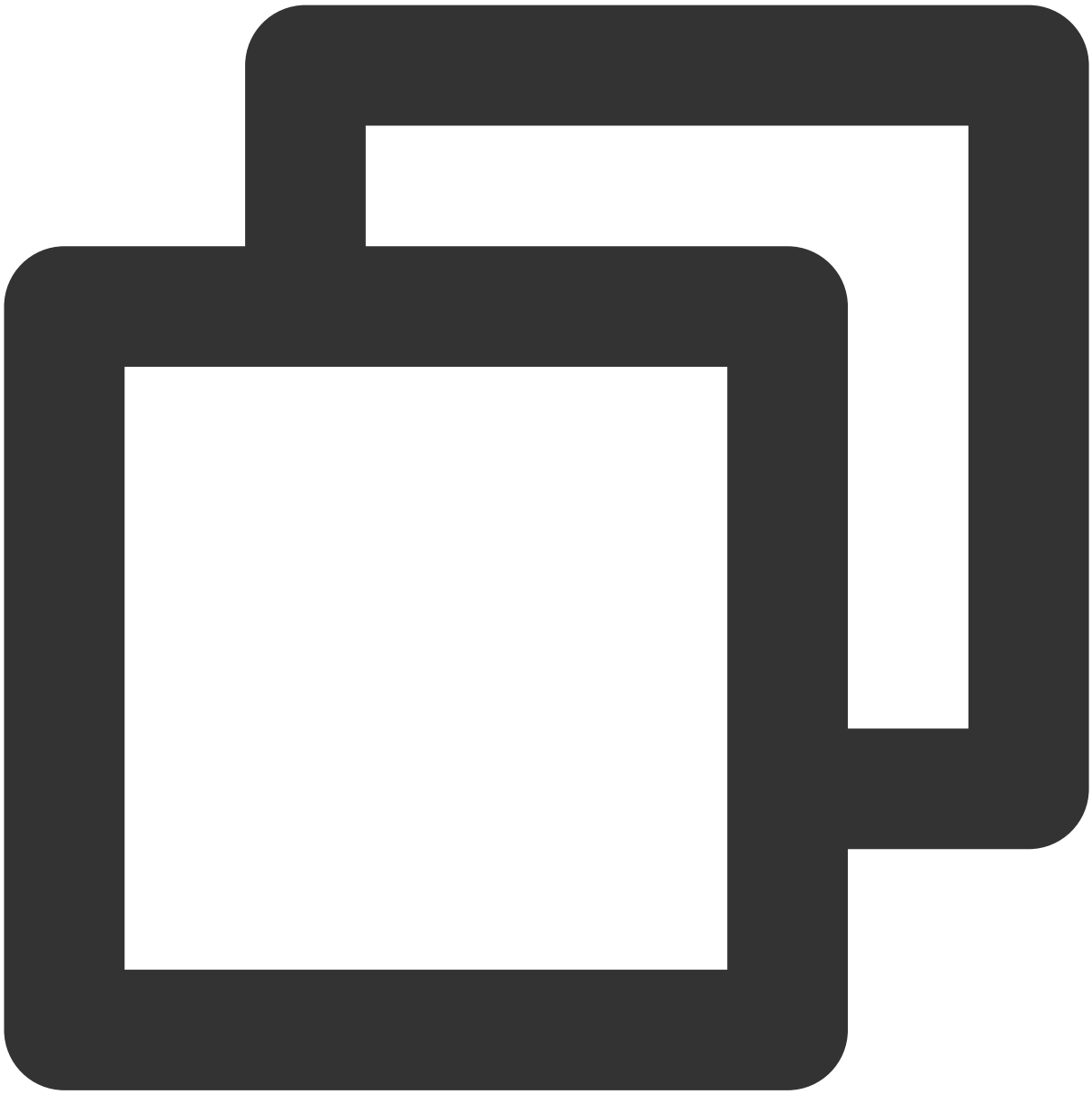
In scenarios where barrage messages need to be displayed, use `TUIBarrageDisplayView` to show the barrage messages:



```
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.



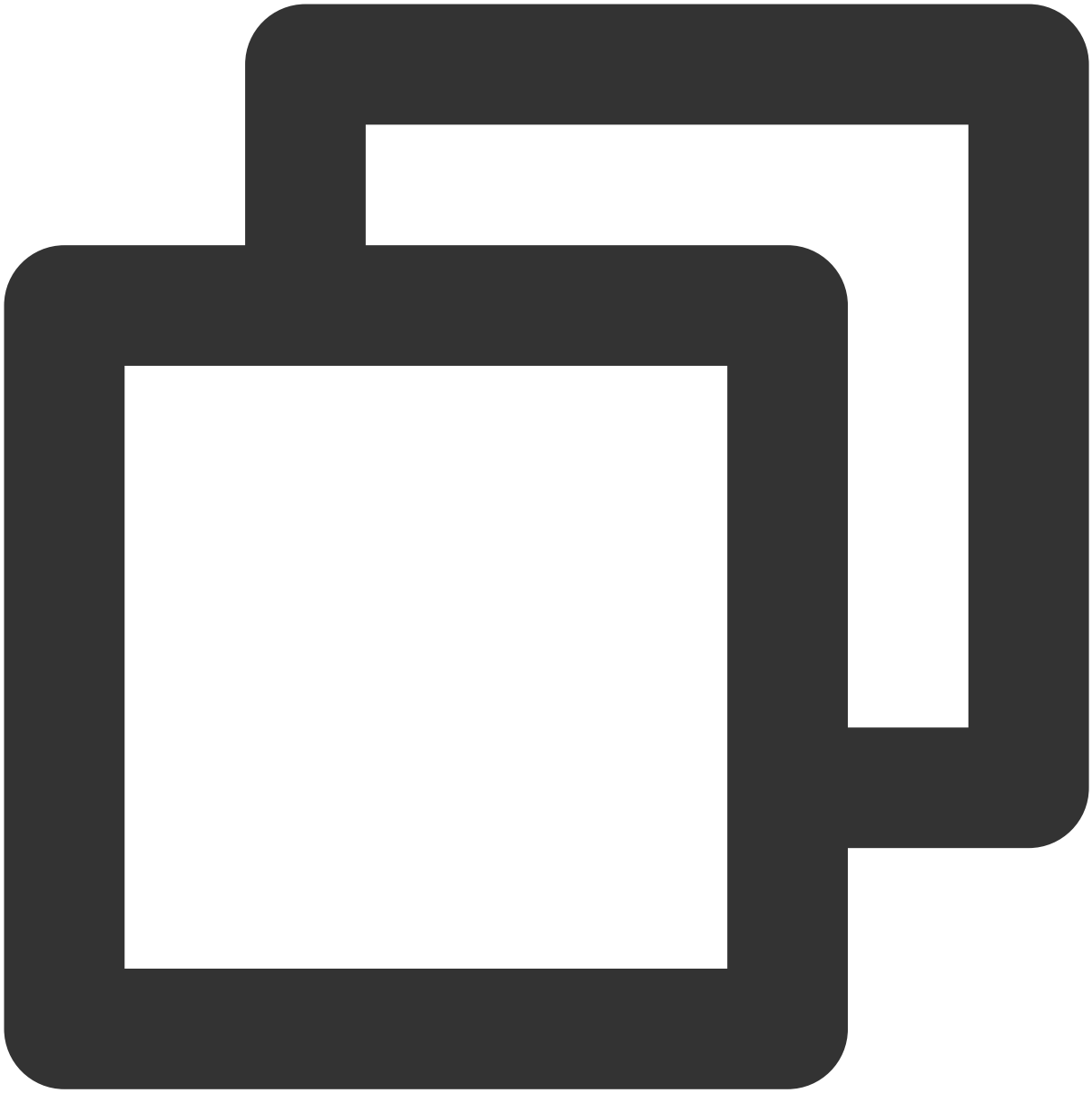
```
barrageDisplayView.delegate = self
extension UIViewController: TUIBarrageDisplayViewDelegate {
   func barrageDisplayView(_ barrageDisplayView: TUIBarrageDisplayView, createCusto
       // Return custom barrage UI here.
    }
}
```

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：

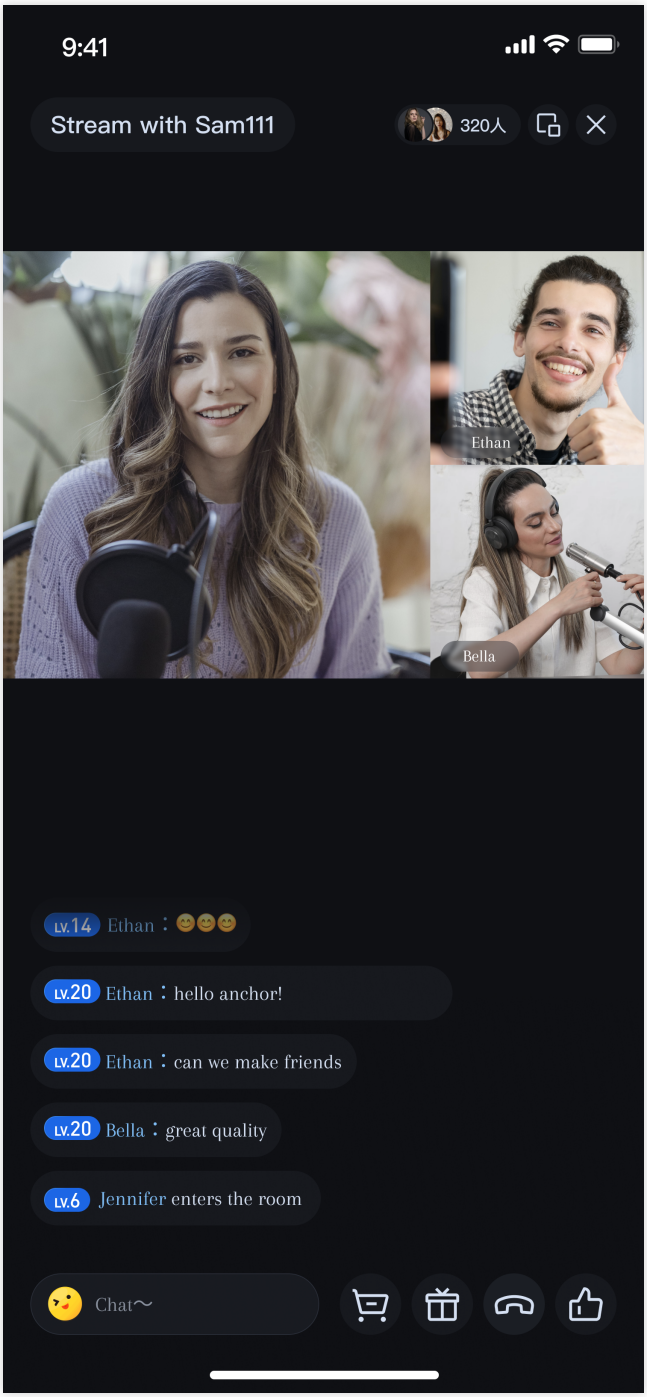
The `extInfo` of `TUIBarrage` is a Map, used for storing custom data.

Android

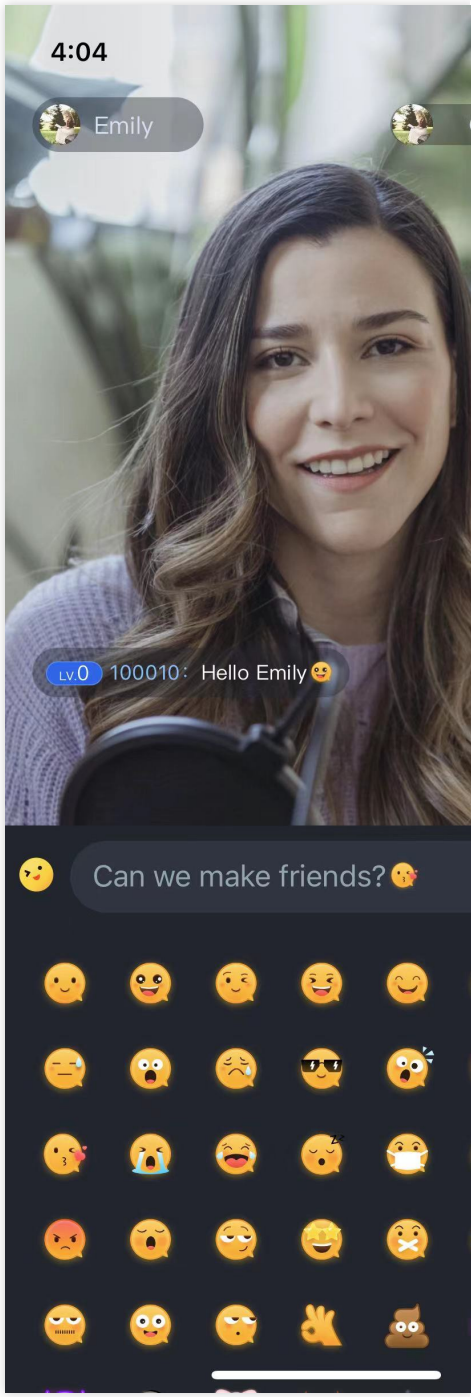
Last updated：2024-08-09 22:25:01

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

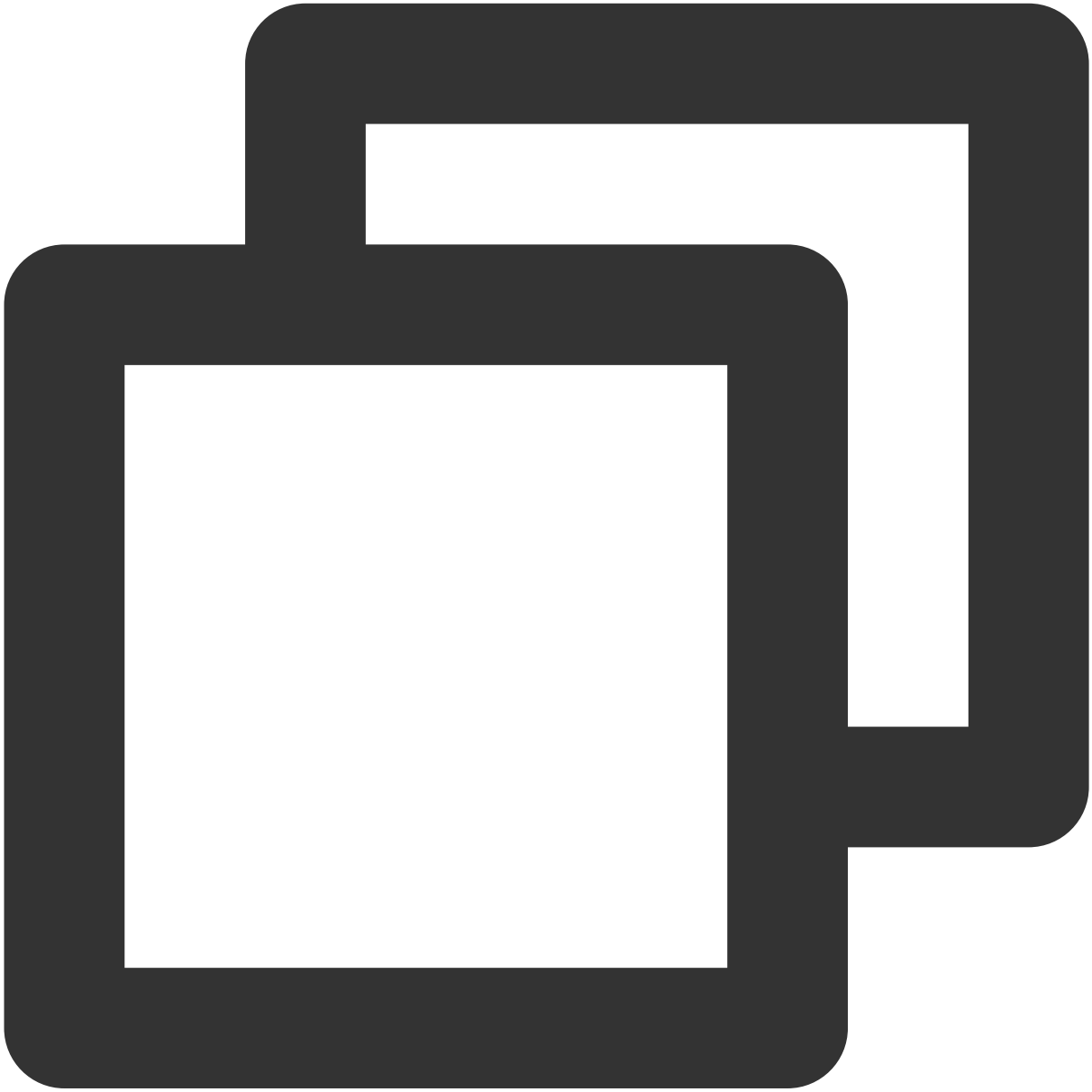
Note：

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

Integration

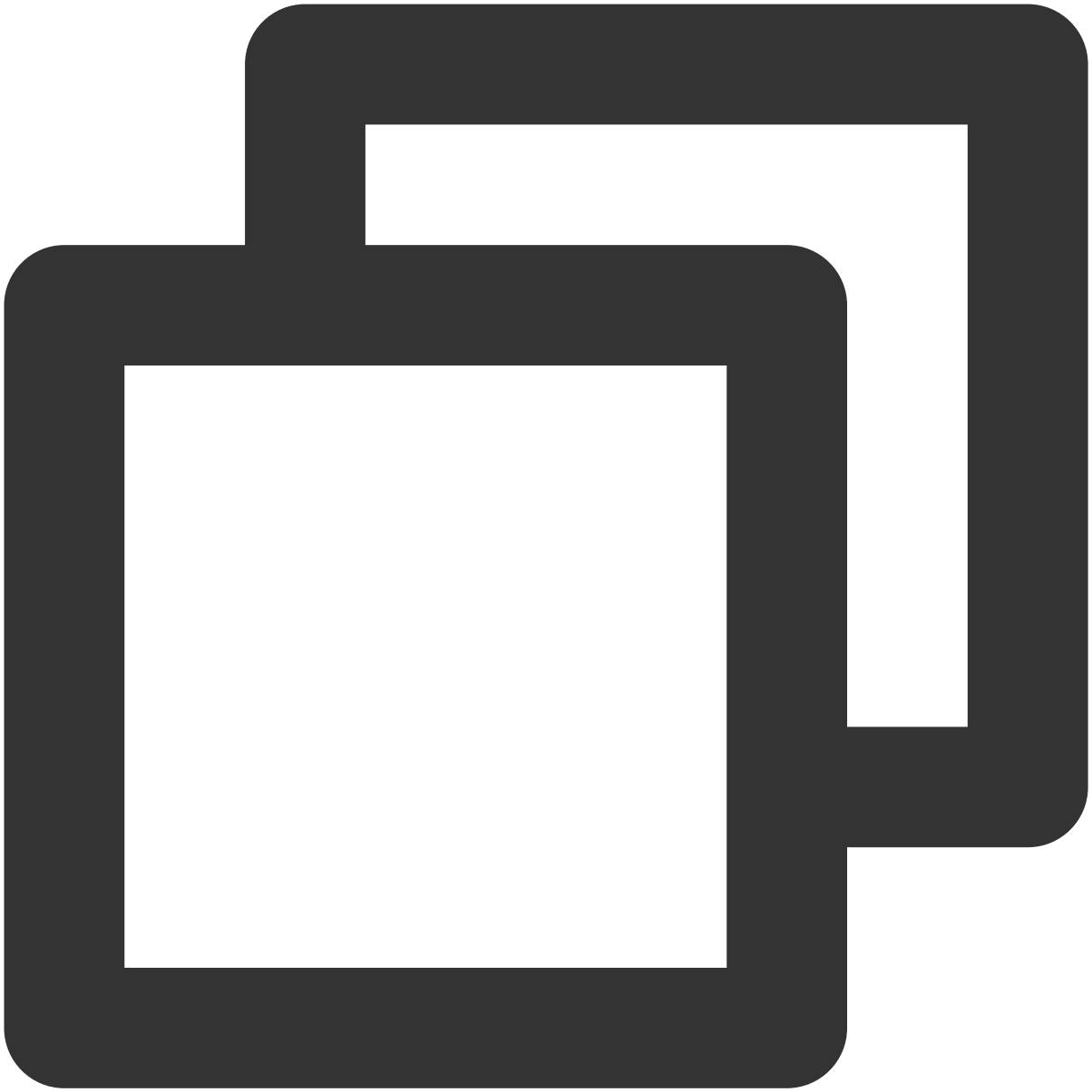
The barrage component mainly provides two Objects：

`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);
```

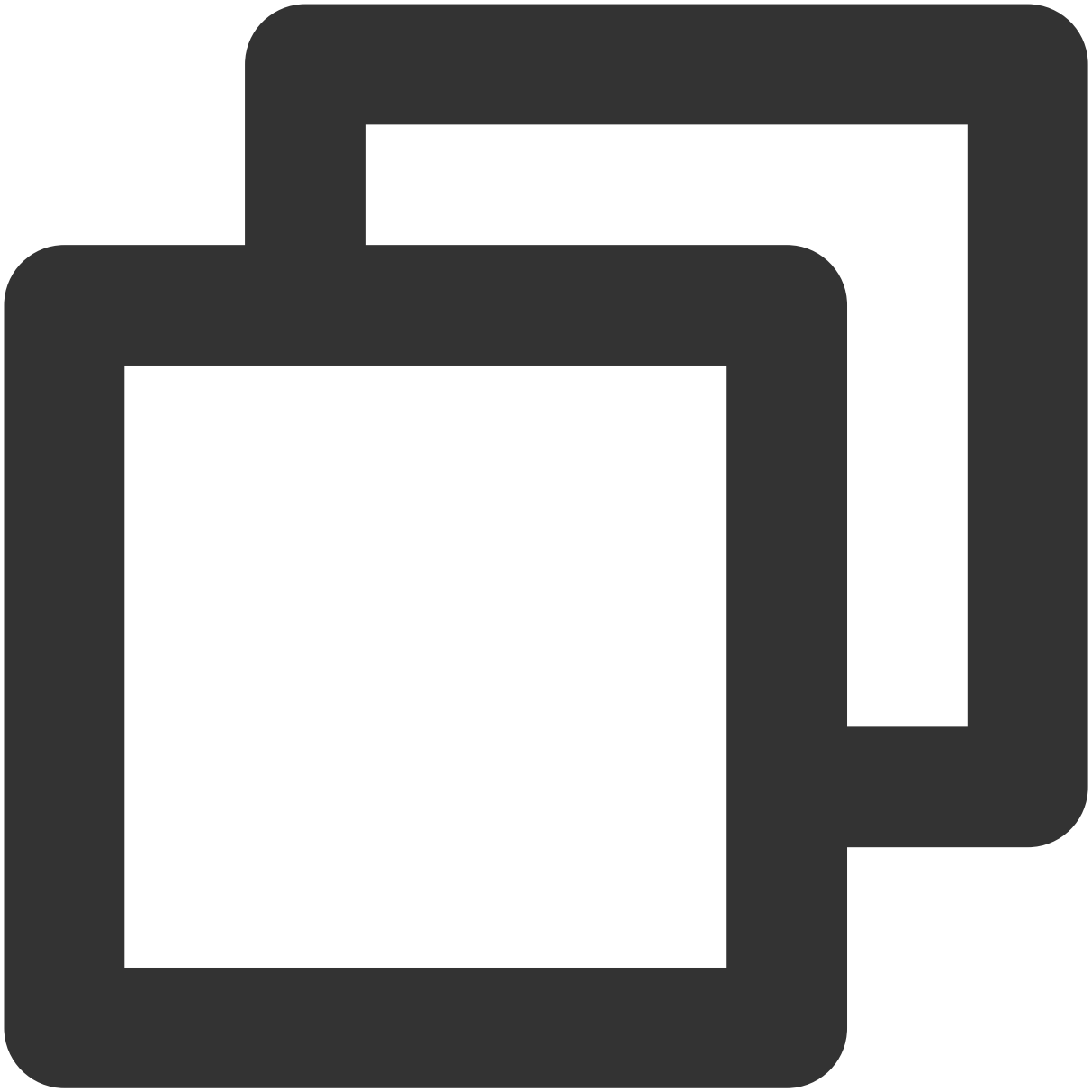
In scenarios where barrage messages need to be displayed, use `TUIBarrageDisplayView` to show the barrage messages:



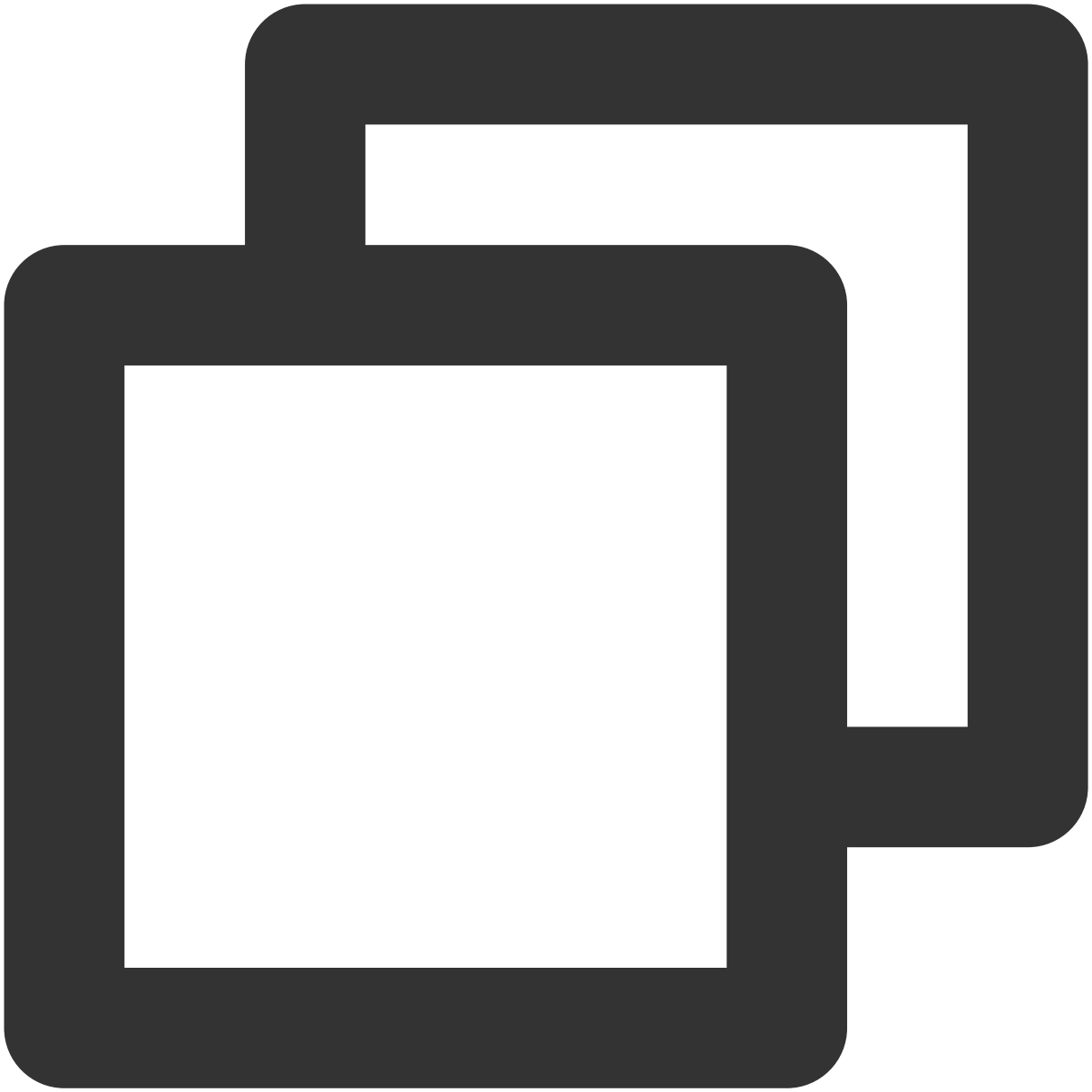
```
TUIBarrageDisplayView barrageDisplayView = new TUIBarrageDisplayView(mContext, room
mLayoutBarrageContainer.addView(barrageDisplayView);
```

Customize message style

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



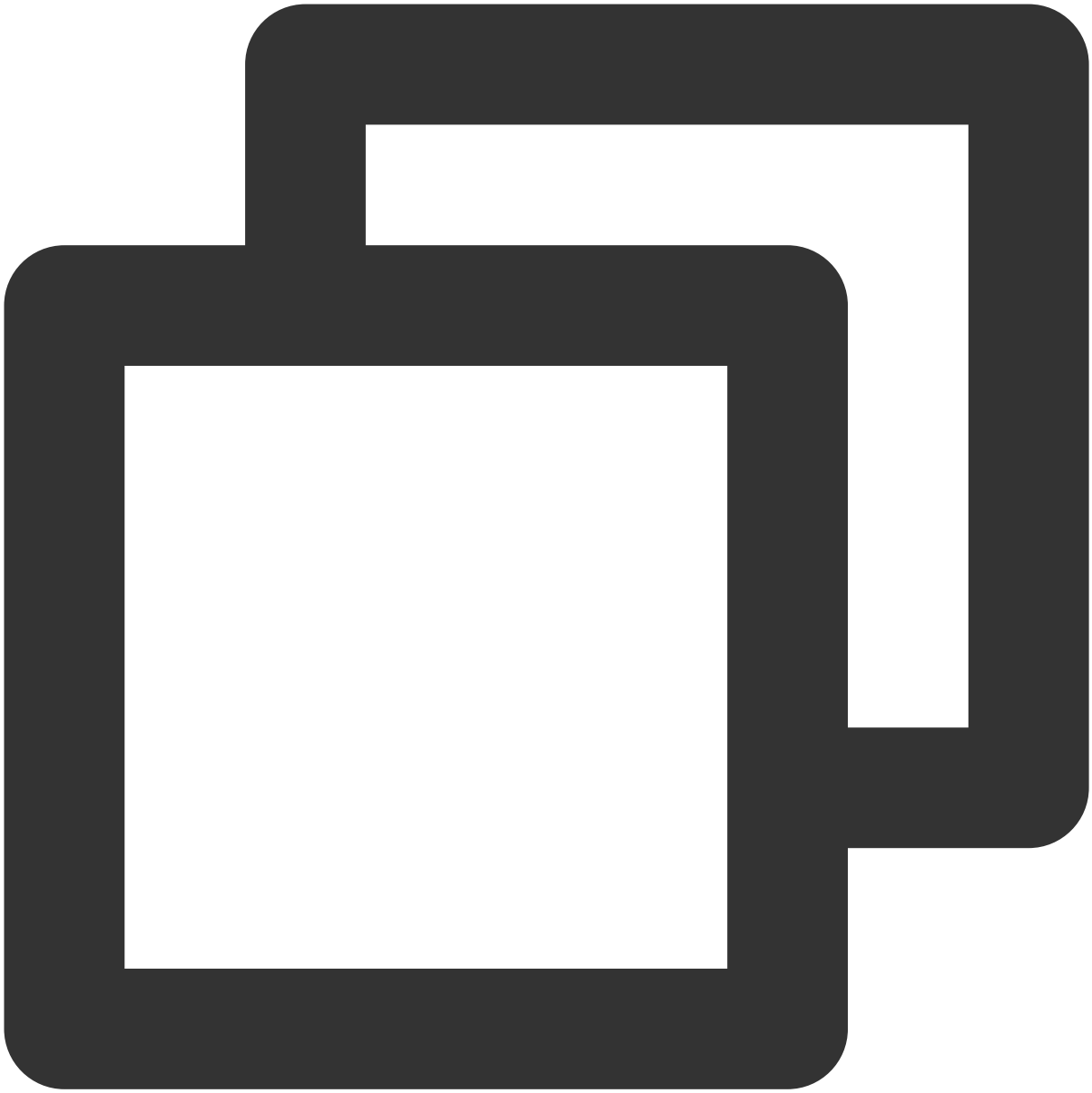
```
public interface TUIBarrageDisplayAdapter {
    RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType);

    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;
```

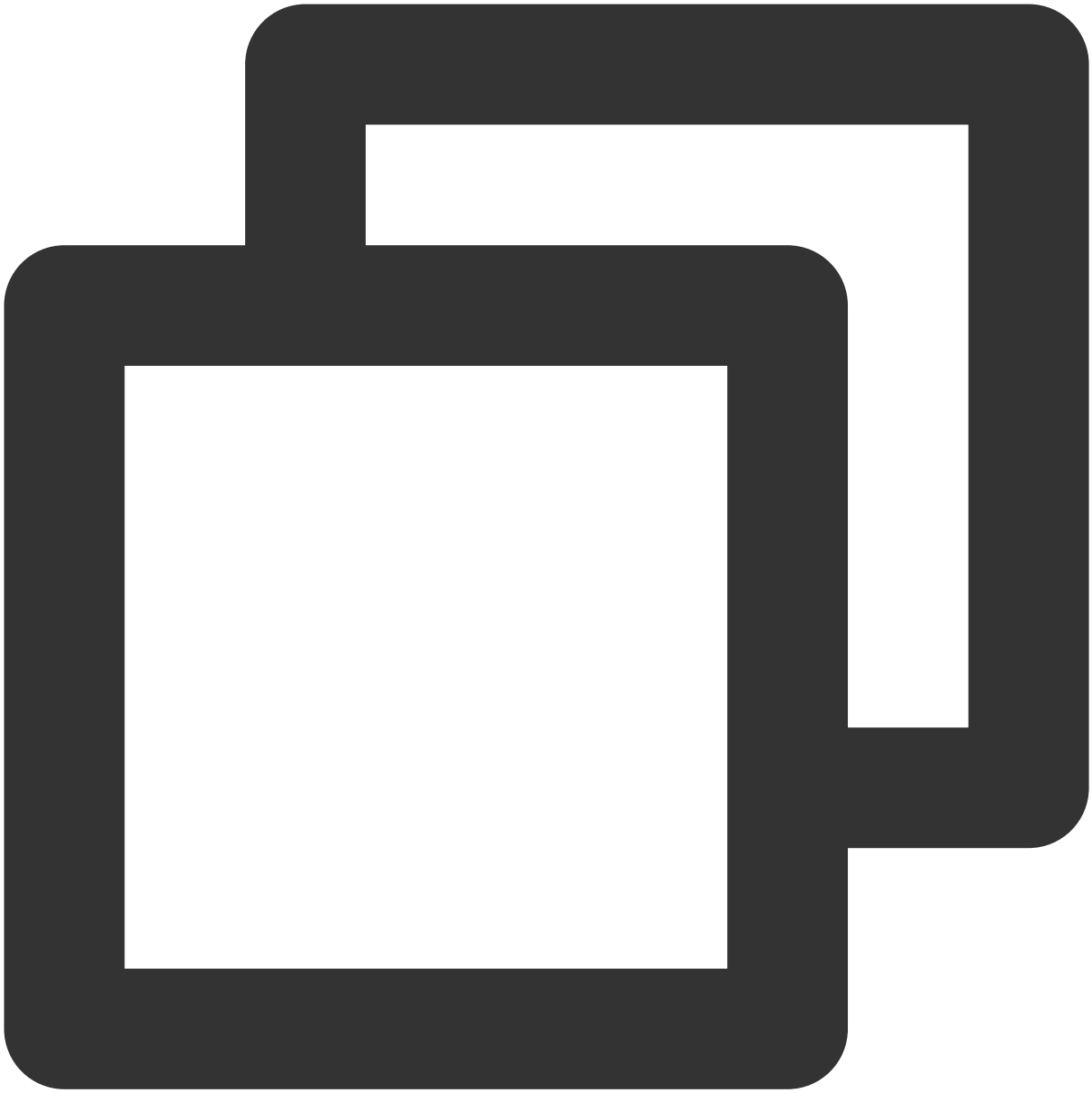
```
    public String level;
}
```

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：

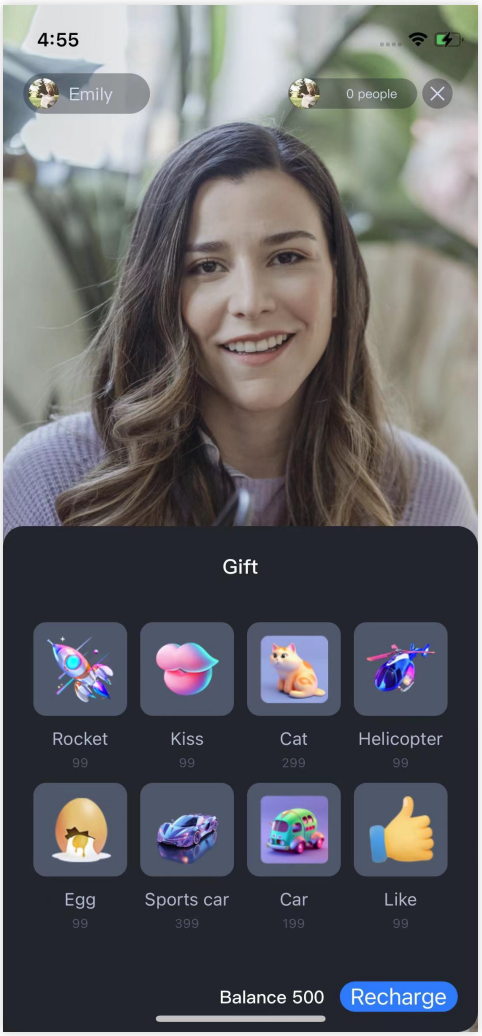
The `extInfo` of `TUIBarrage` is a Map, used for storing custom data.

Interactive Gifts (TUILiveKit)
iOS

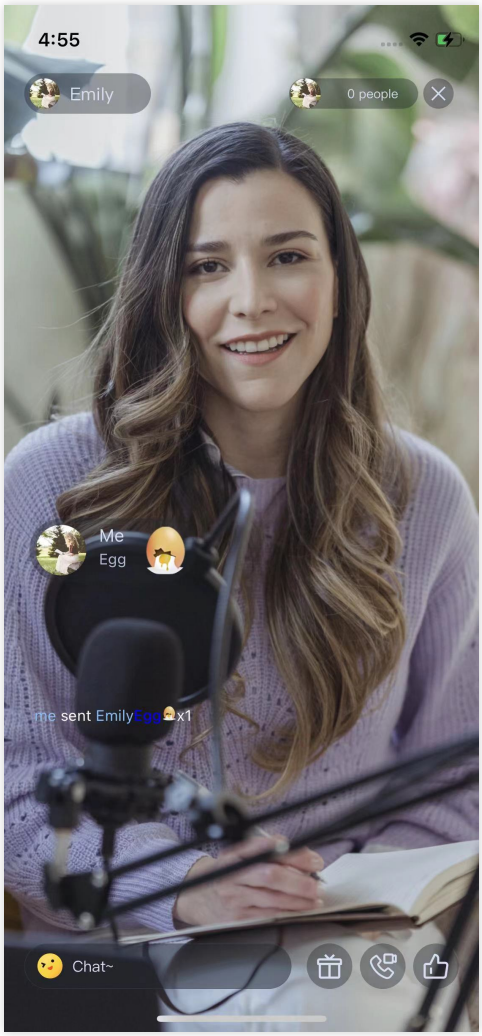
Last updated：2024-07-10 16:31:15

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

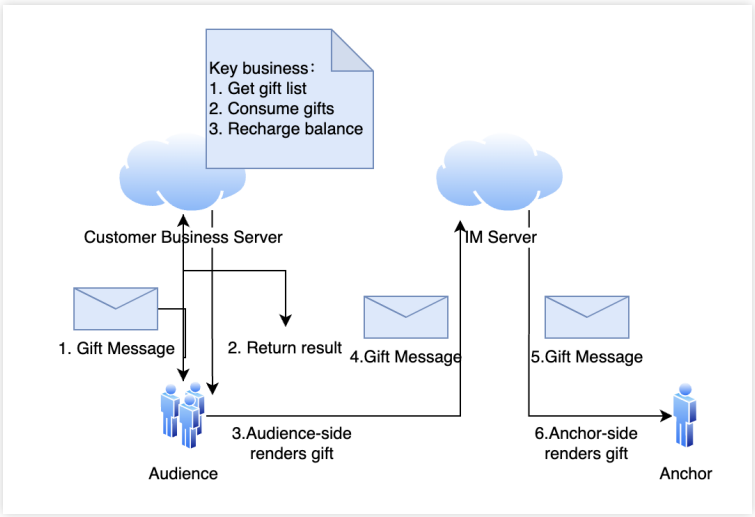
Pl

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

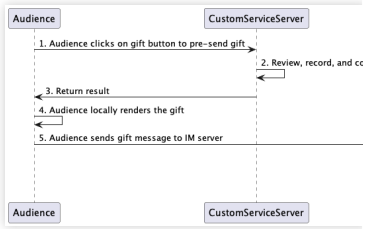
1. After billing, the sender directly sees that XXX sent XXX gifts (to ensure that the sender sees the gifts he sent, and the abandonment policy may be triggered when the message volume is large).

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

Gift System



Gift system`s Structure diagram



Gift system`s Sequence diagram

Integration

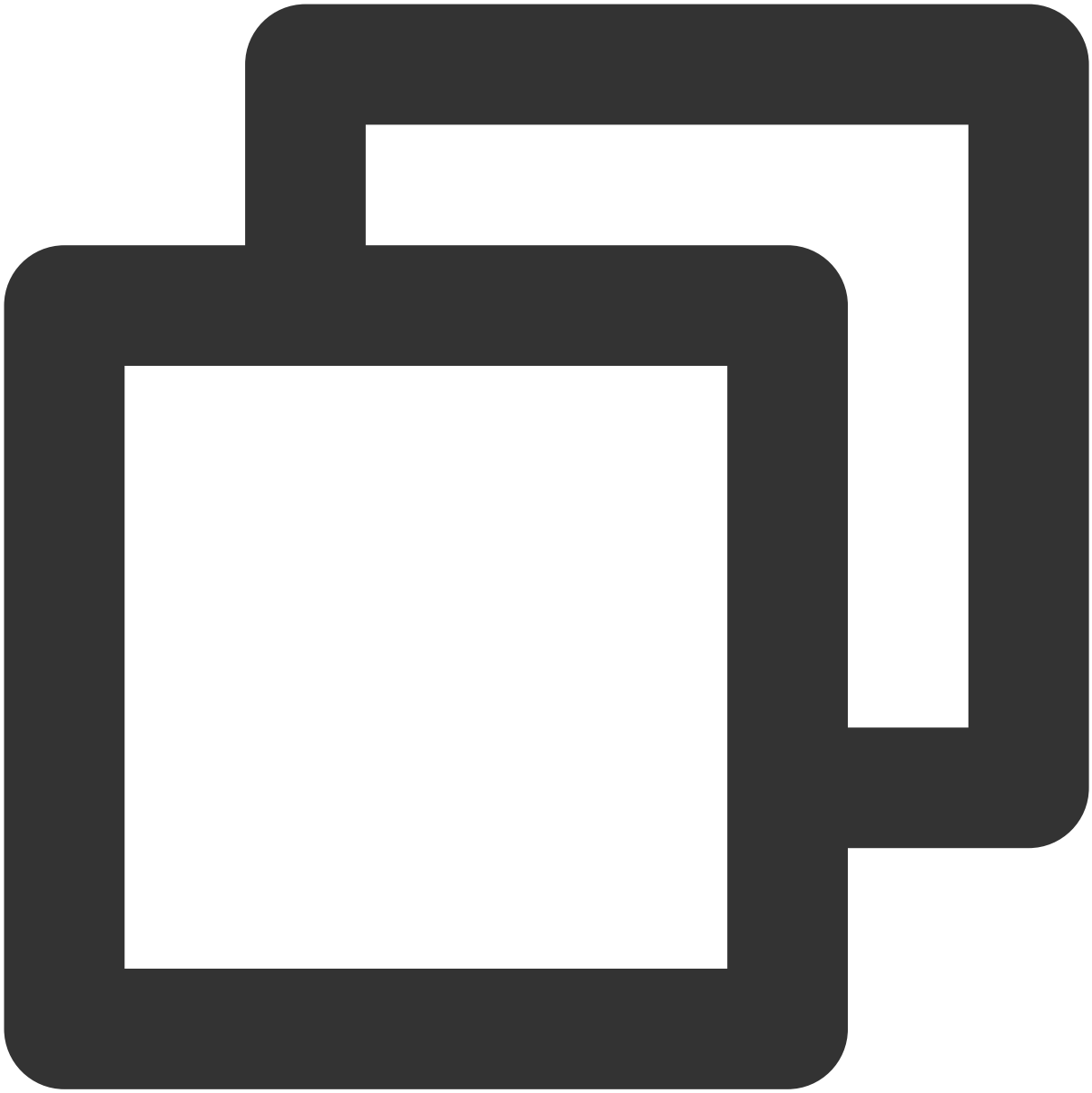
The gift component mainly provides 2 objects:

`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

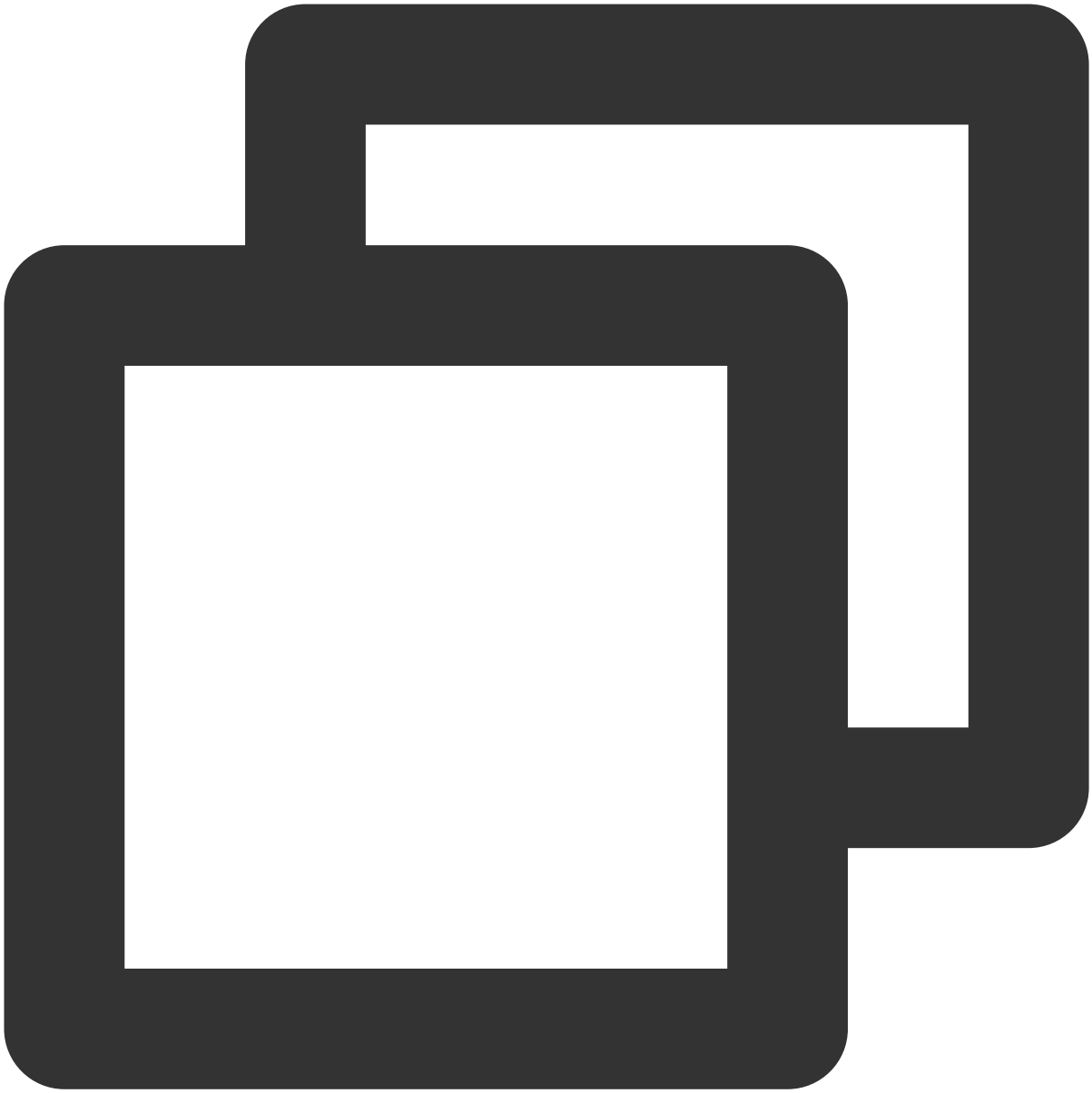
`giftName: String` : Gift Name

`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

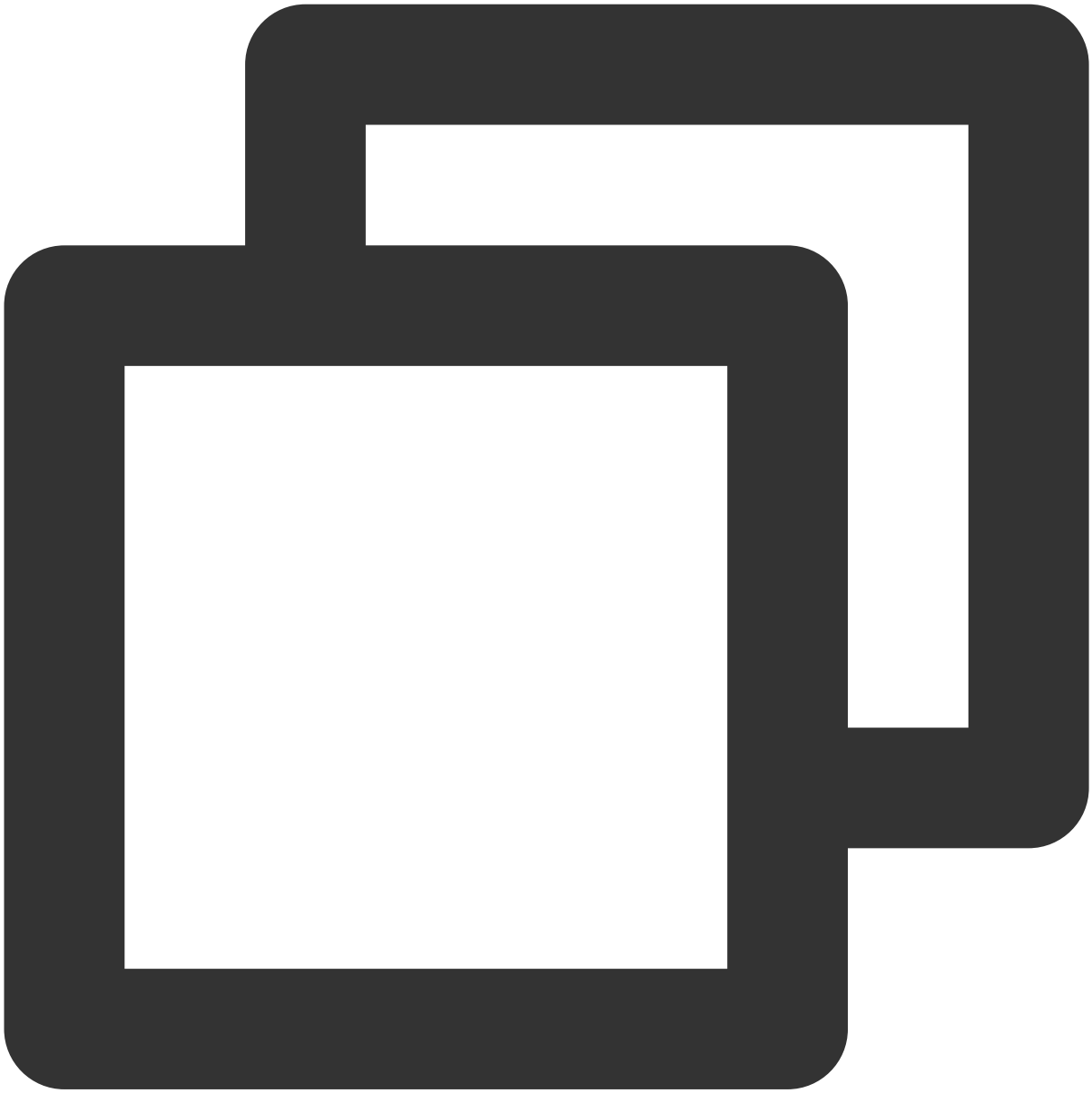
Implement the `onSendGift` delegate function in the `TUIGiftListViewDelegate` of `TUIGiftListView` to get the gift count and gift information. After preprocessing, you can call the `sendGift` function of `TUIGiftListView` for the actual sending of gifts.



```
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.

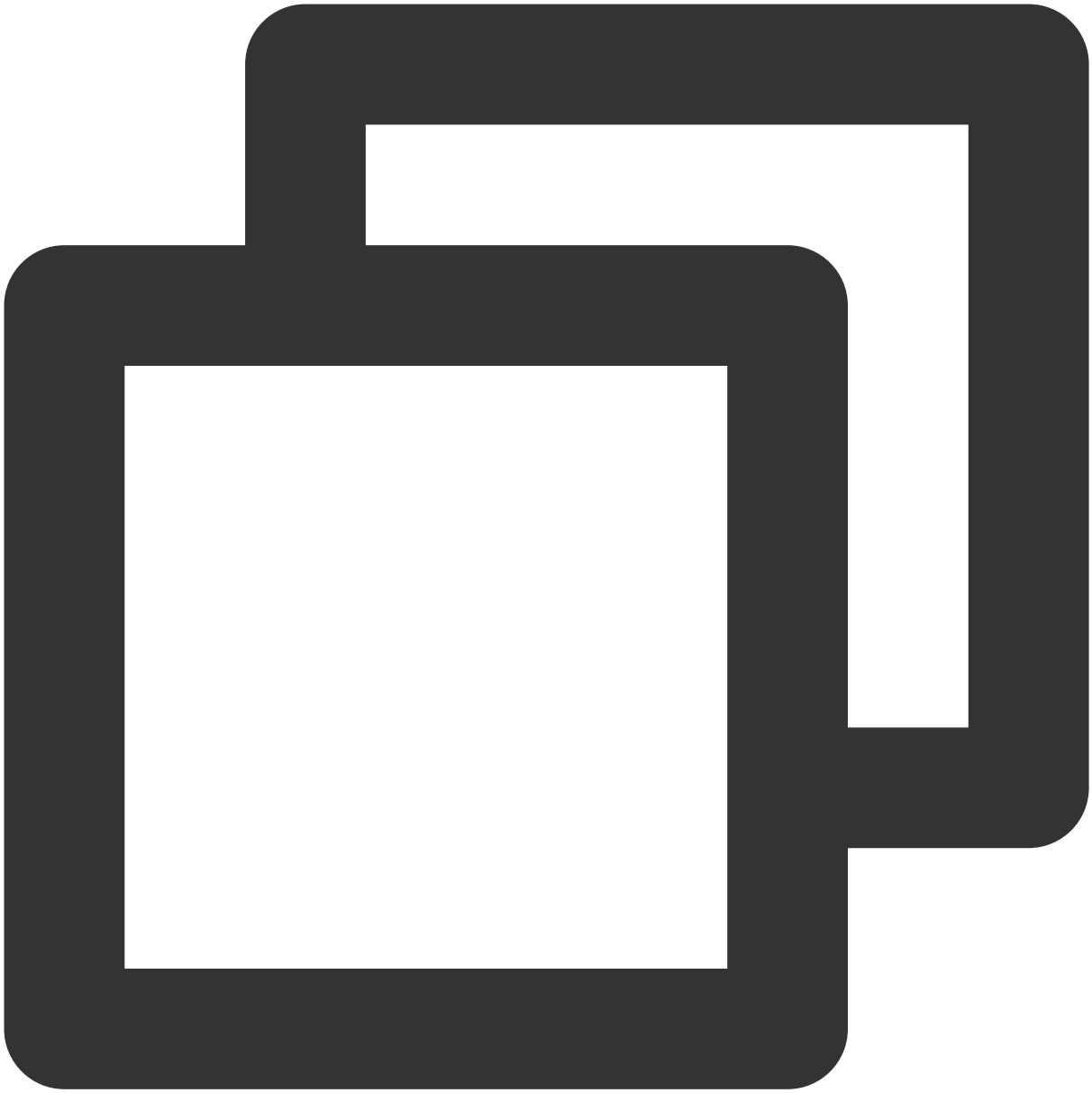


```
let giftDisplayView: TUIGiftPlayView = TUIGiftPlayView(groupId:xxx)
```

Note：

`TUIGiftPlayView` requires full-screen integration.

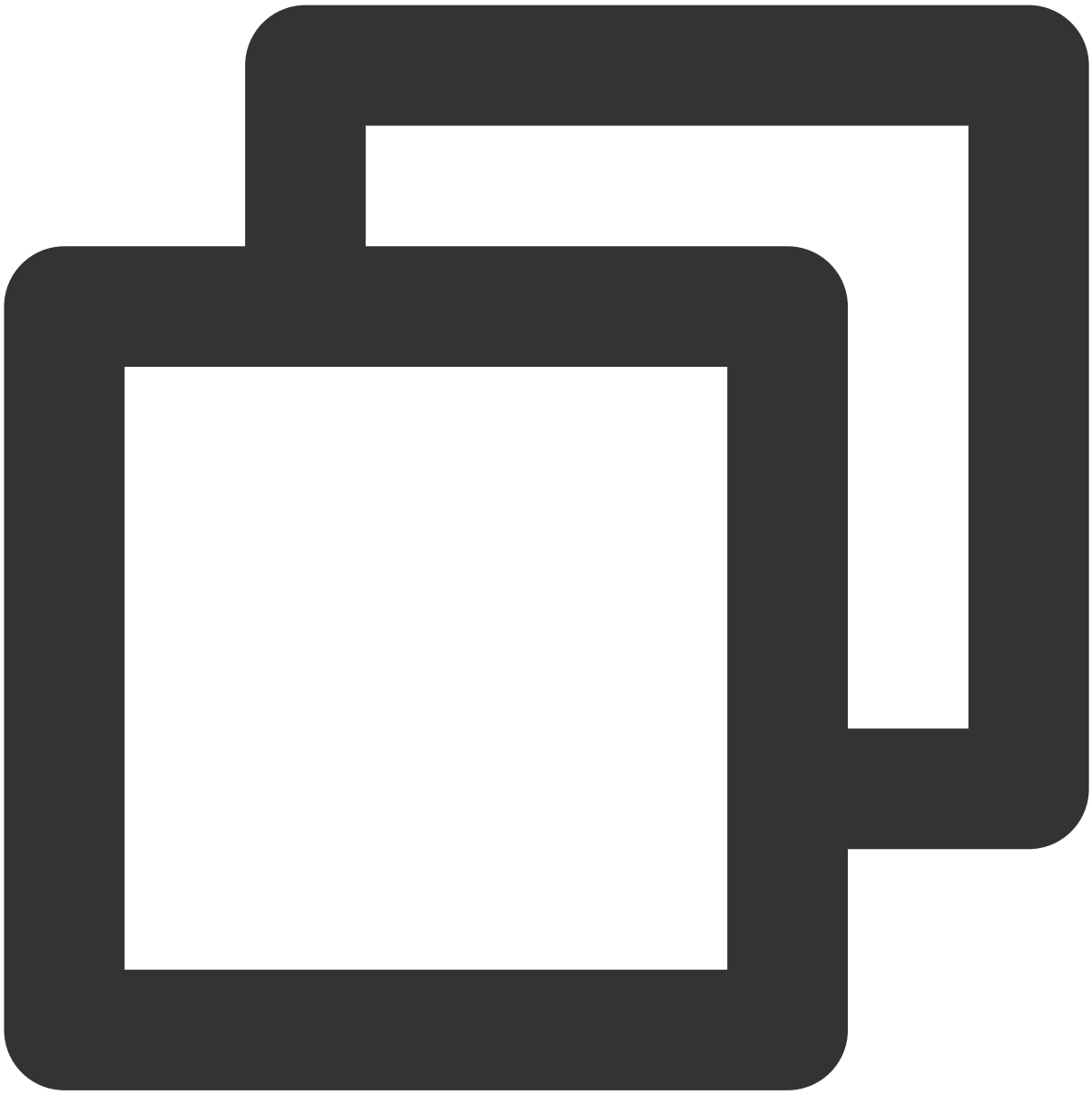
If you need to get the callback information of receiving gifts, you can implement the `giftPlayView:onPlayGift` delegate function in the `TUIGiftPlayViewDelegate` of `TUIGiftPlayView` .



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

Play Gift Animation

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



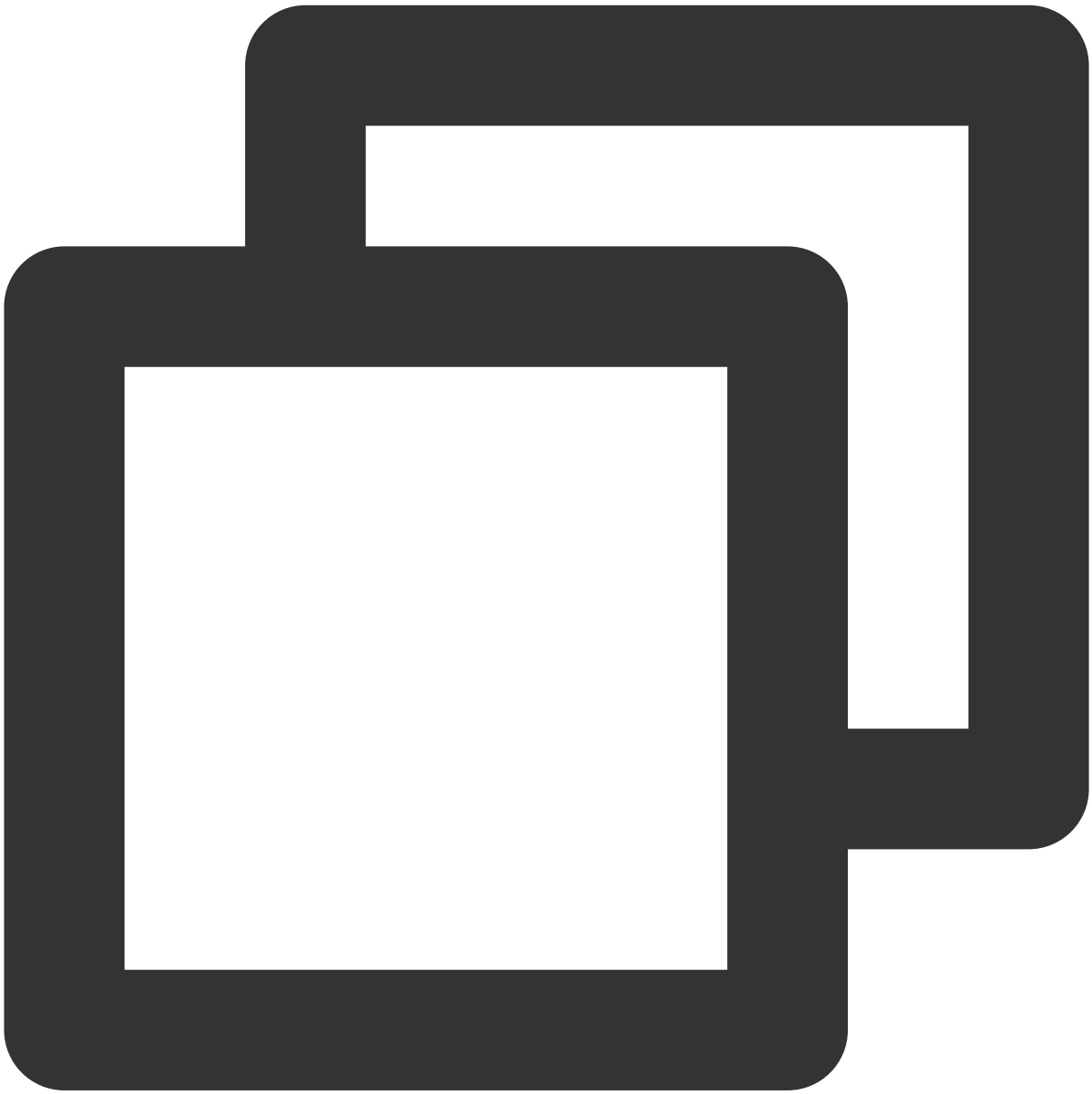
```
extension ViewController: TUIGiftPlayViewDelegate {
    func giftPlayView(_ giftPlayView: TUIGiftPlayView, onPlayGiftAnimation gift: TU
      //...
    }
}
```

Note：

Only SVGA animations are supported.

Set balance

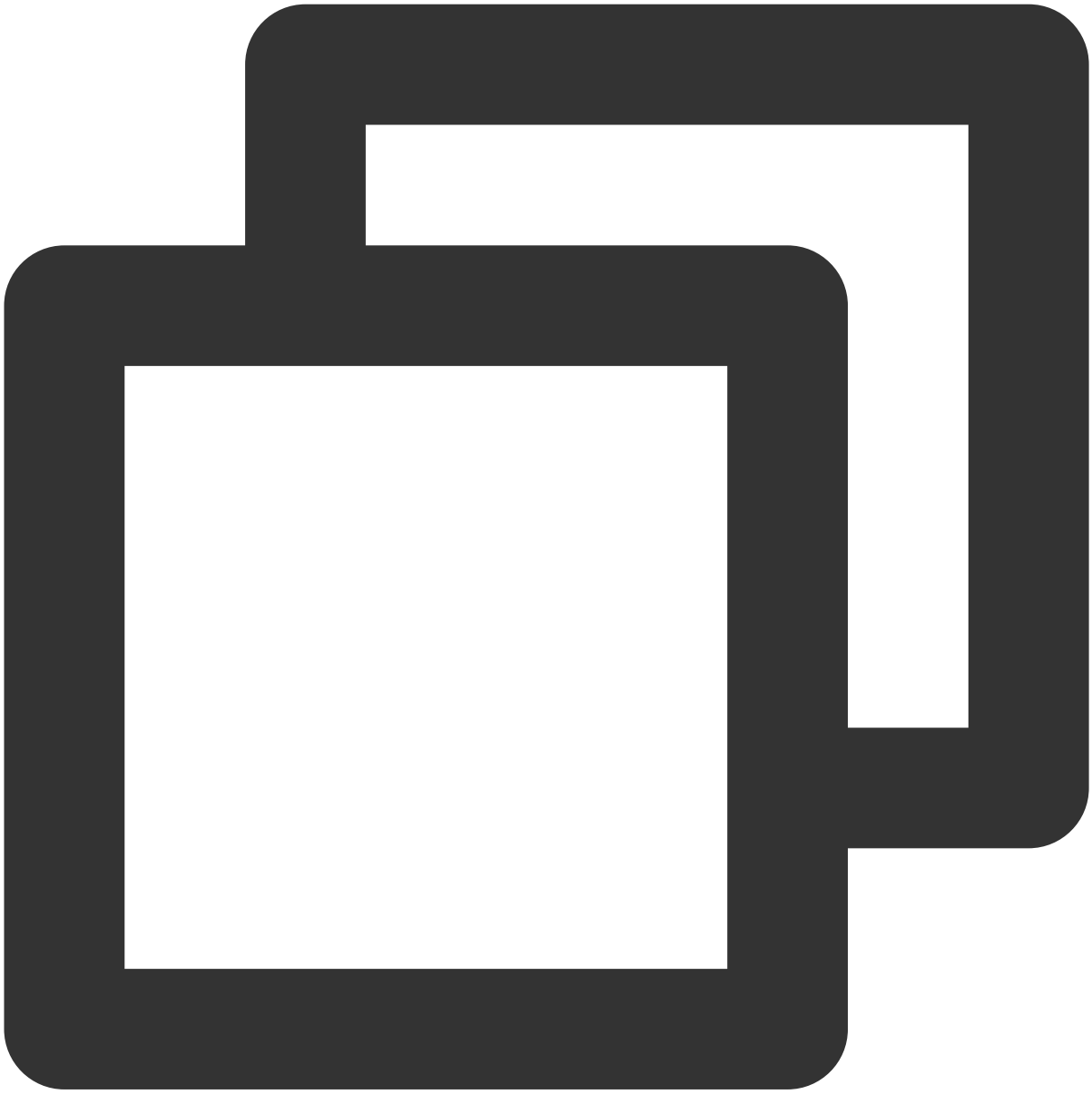
The gift panel component `TUIGiftListView` provides the `setBalance` interface, which can be used to set the balance value displayed on the gift panel.



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

Recharge

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.



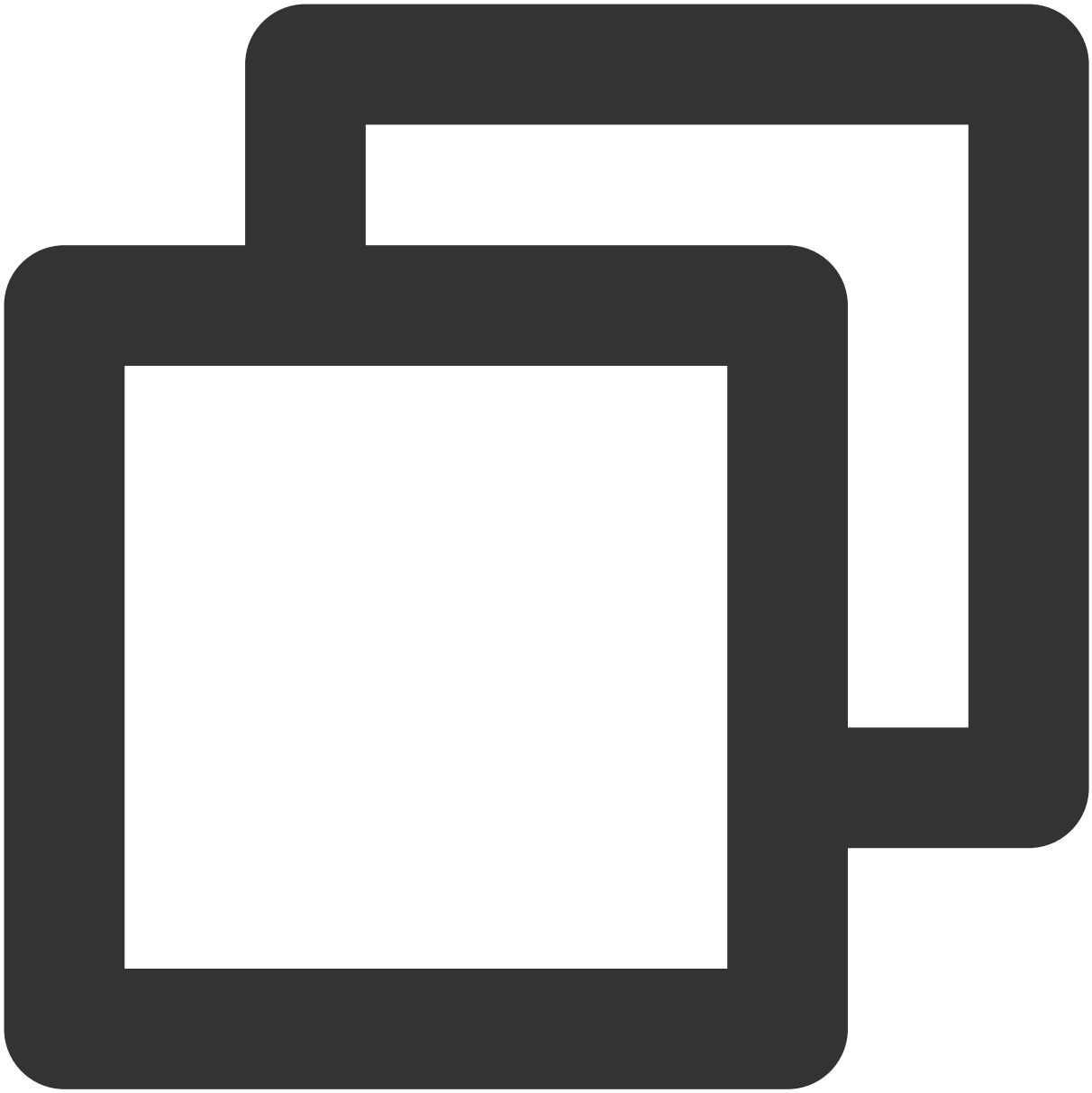
```
extension ViewController: TUIGiftListViewDelegate {
    func onRecharge(giftListView view: TUIGiftListView) {
        //...This can be used to connect to your own recharge system. After the rec
        //you can call view.setBalance(xxx) to set the balance display of the gift
    }
}
```

Note：

1. The gift balance is a concept of virtual currency, not real money.
2. 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

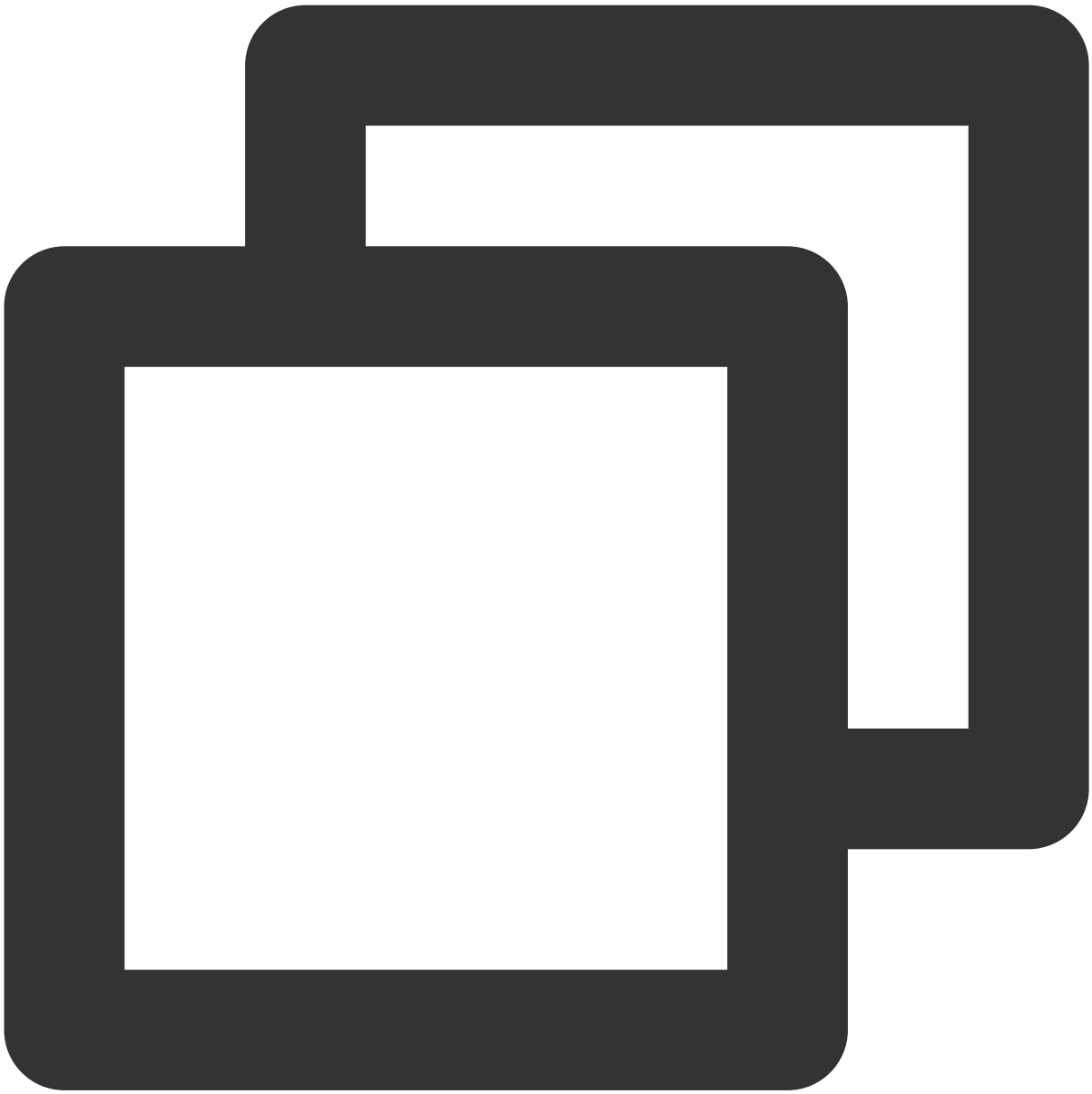
Implement the `onSendGift` delegate function in the `TUIGiftListViewDelegate` of `TUIGiftListView`, connect to the customer's own business server, complete the balance verification, gift billing, and consumption statistics, and then call the `sendGift` of `TUIGiftListView` to send the gift message.



```
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 {
```

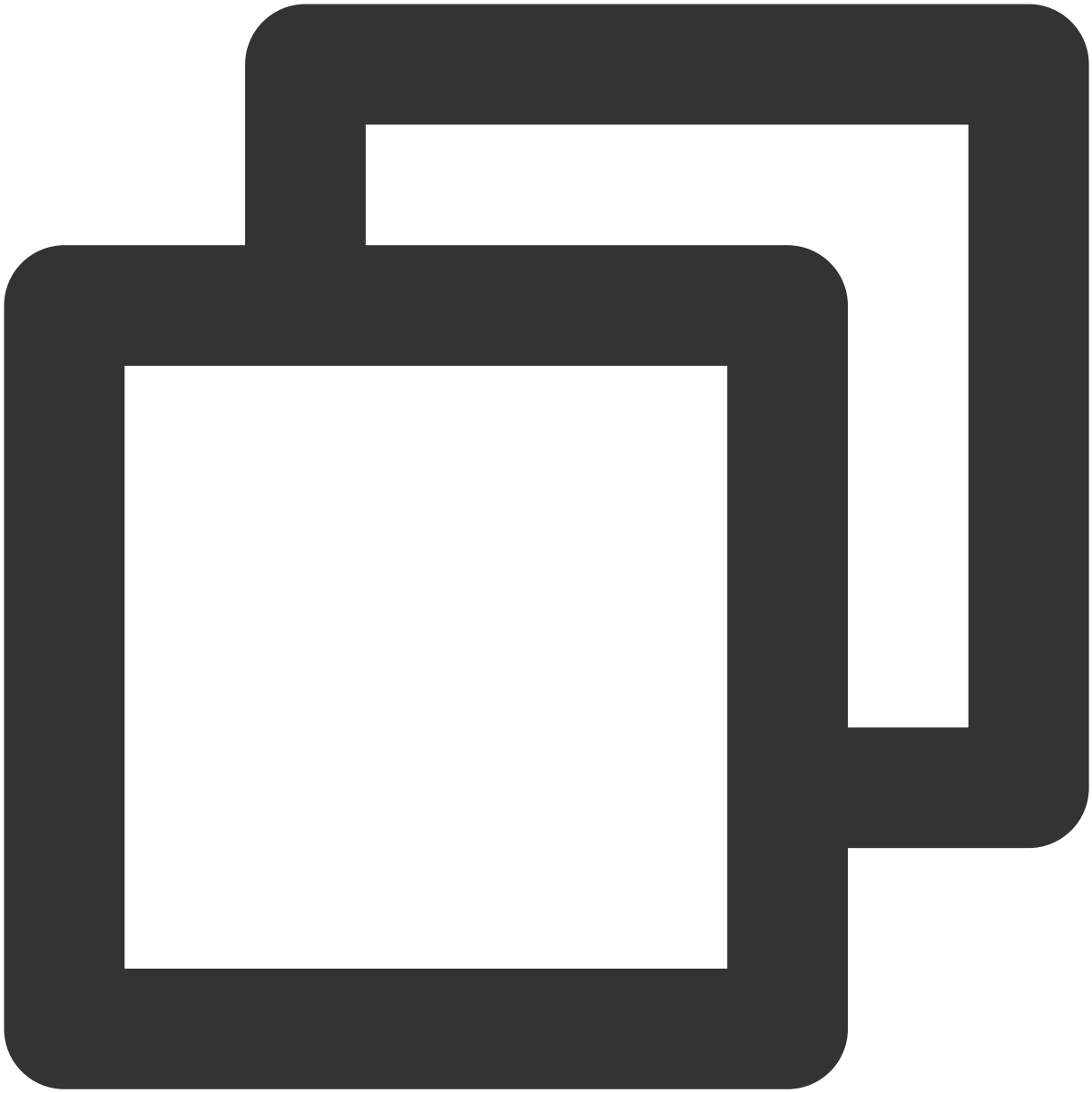
```
            if error == .noError {
                view.setGiftList(giftList)
            } else {
                self.makeToast("query gift list error, code = \\(error)")
            }
        }
    }
    return view
}()
```

Note：

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

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

Customize giftPanel`s balance



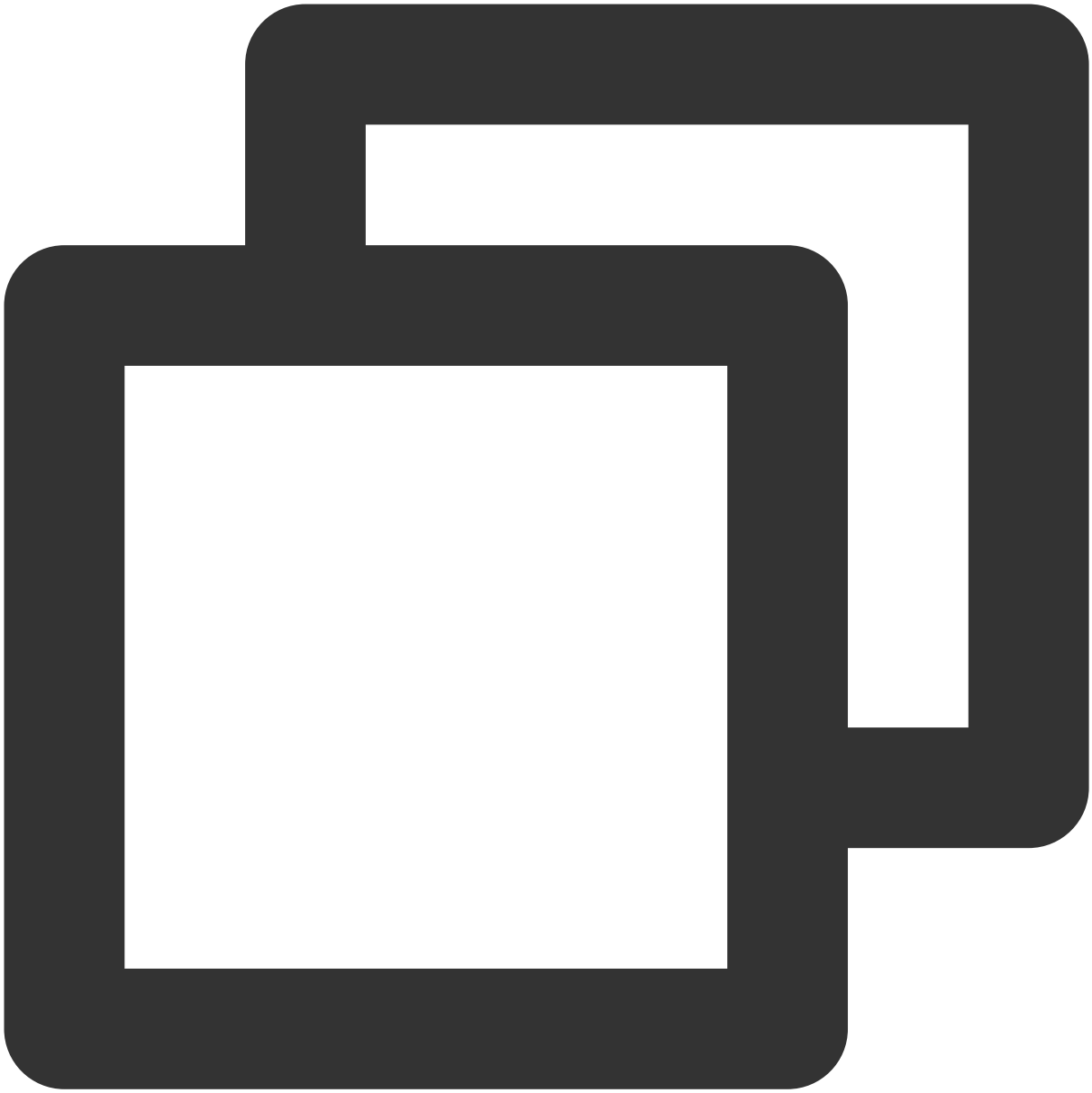
```
// Source code path：TUILiveKit/Source/LiveRoom/View/Audience/Component/AudienceLiv

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



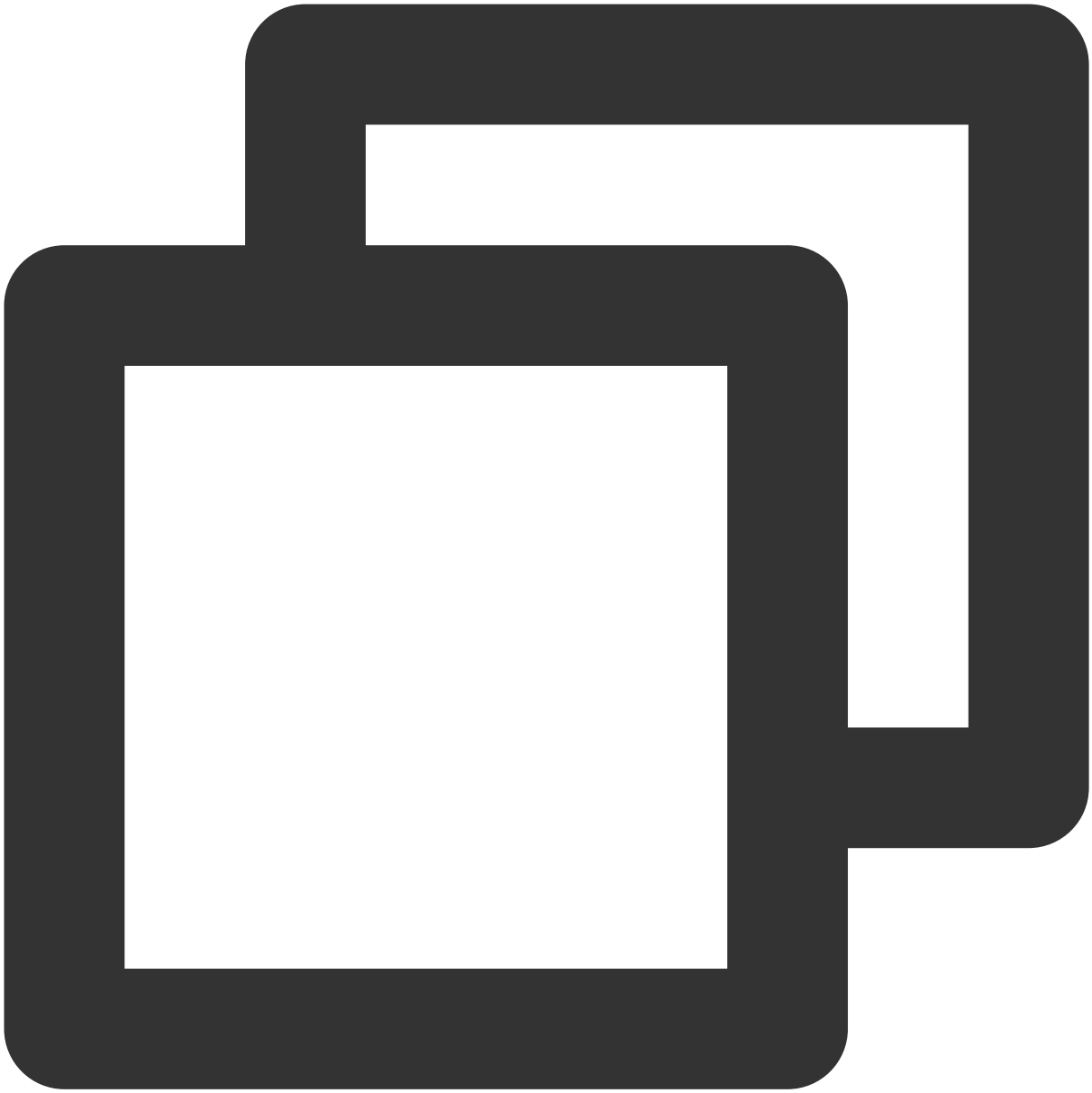
```
// Source code path：TUILiveKit/Source/LiveRoom/View/Audience/Component/AudienceLiv

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 load and play gift animation



```
// 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：

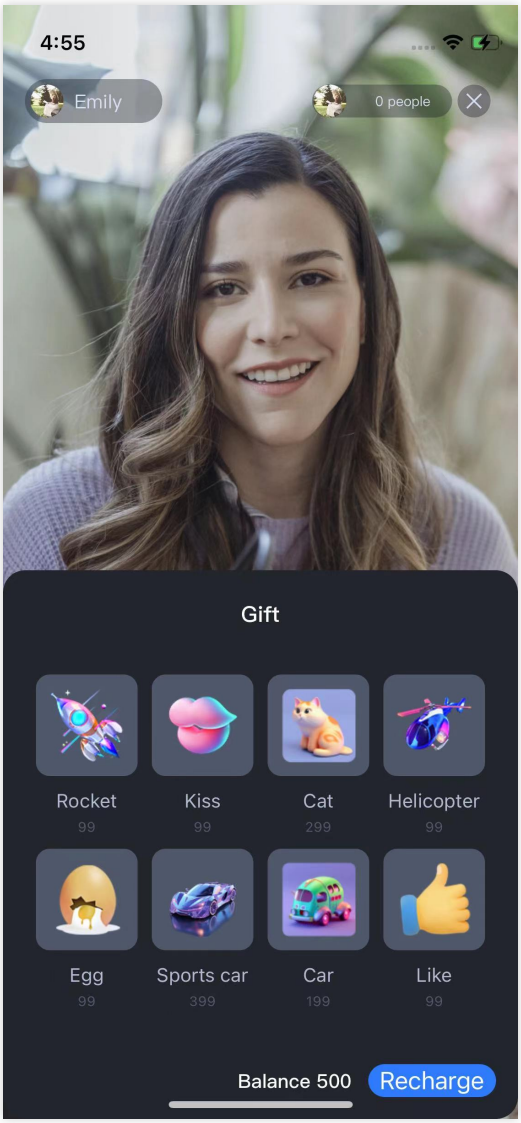
Customers implement the logic of `giftCacheService.request` on their own, successfully load the animation to get the `data` (of `Data` type), and then play the gift animation through `playGiftAnimation` of `TUIGiftPlayView`.

Android

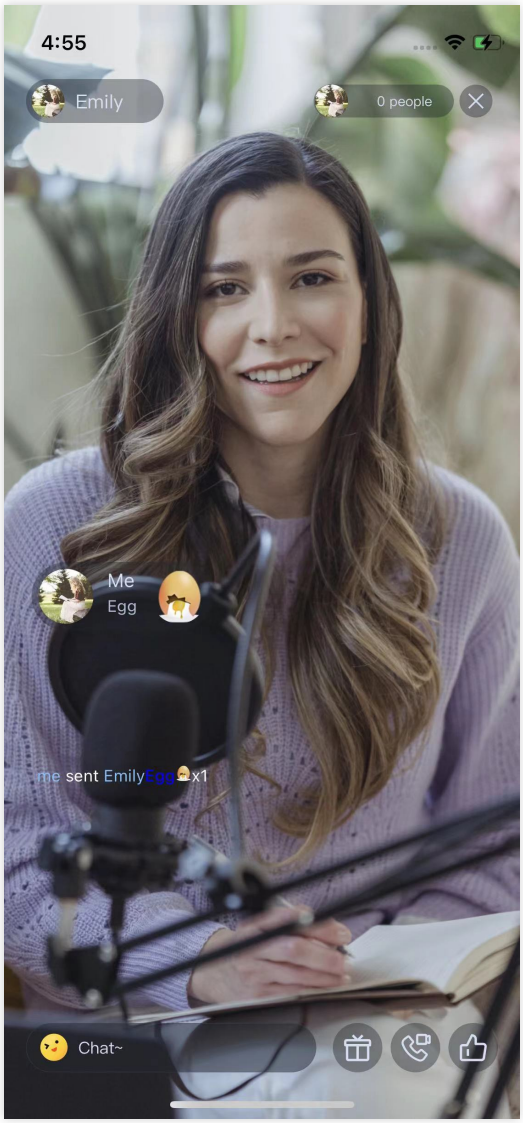
Last updated：2024-07-10 16:31:15

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

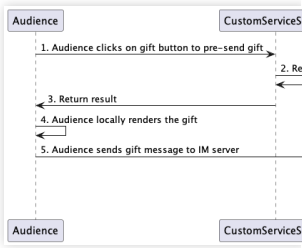


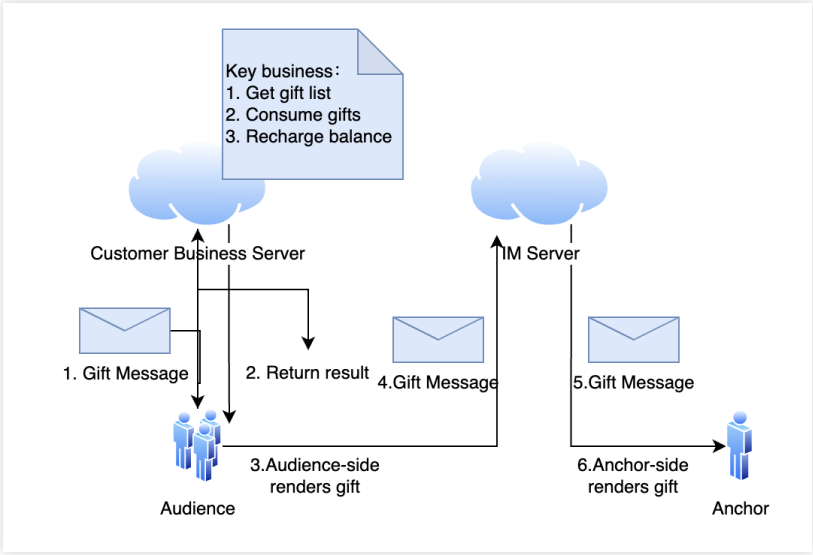
Display Gifts



Play normal gift

Gift System





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

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

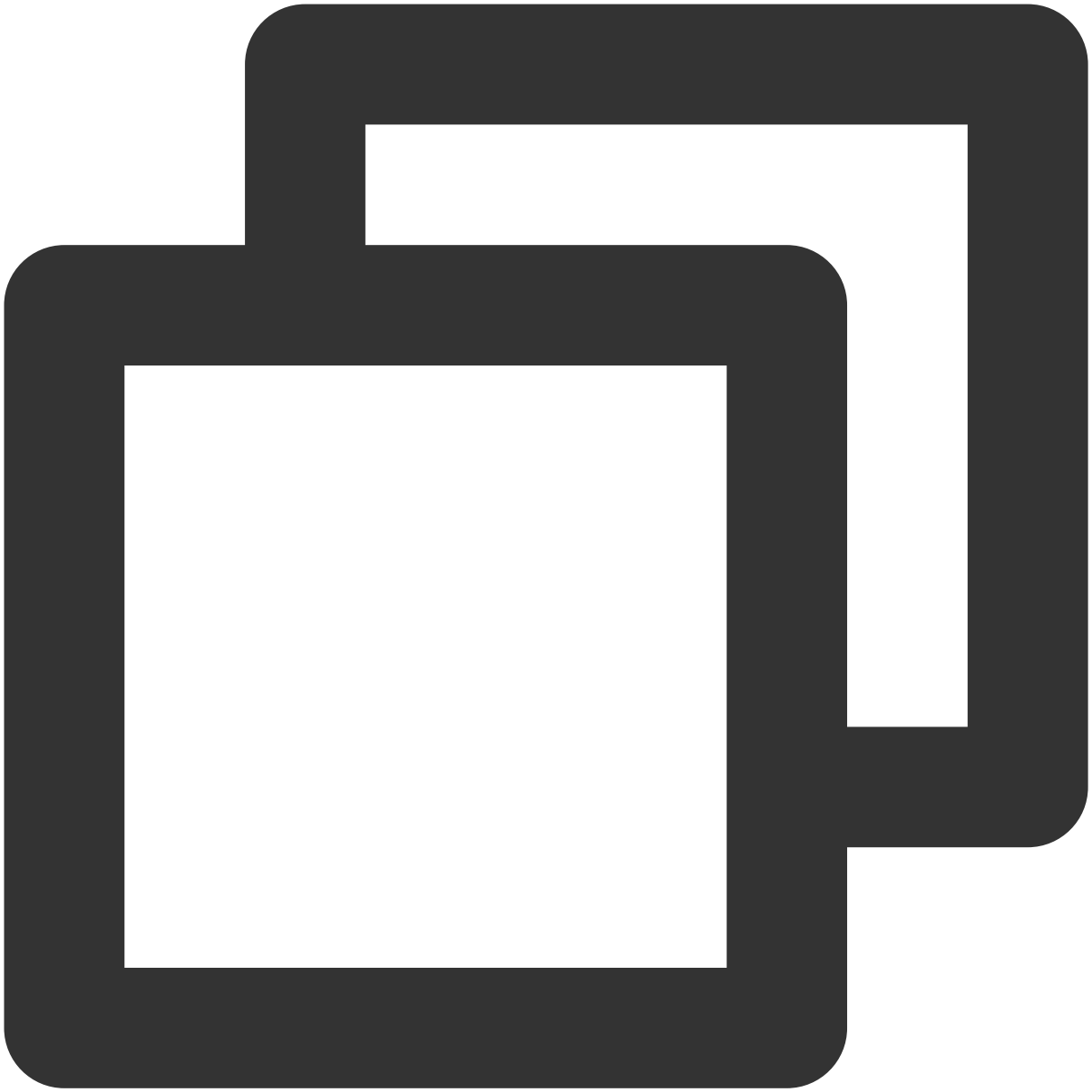
The gift component mainly provides 2 objects:

`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

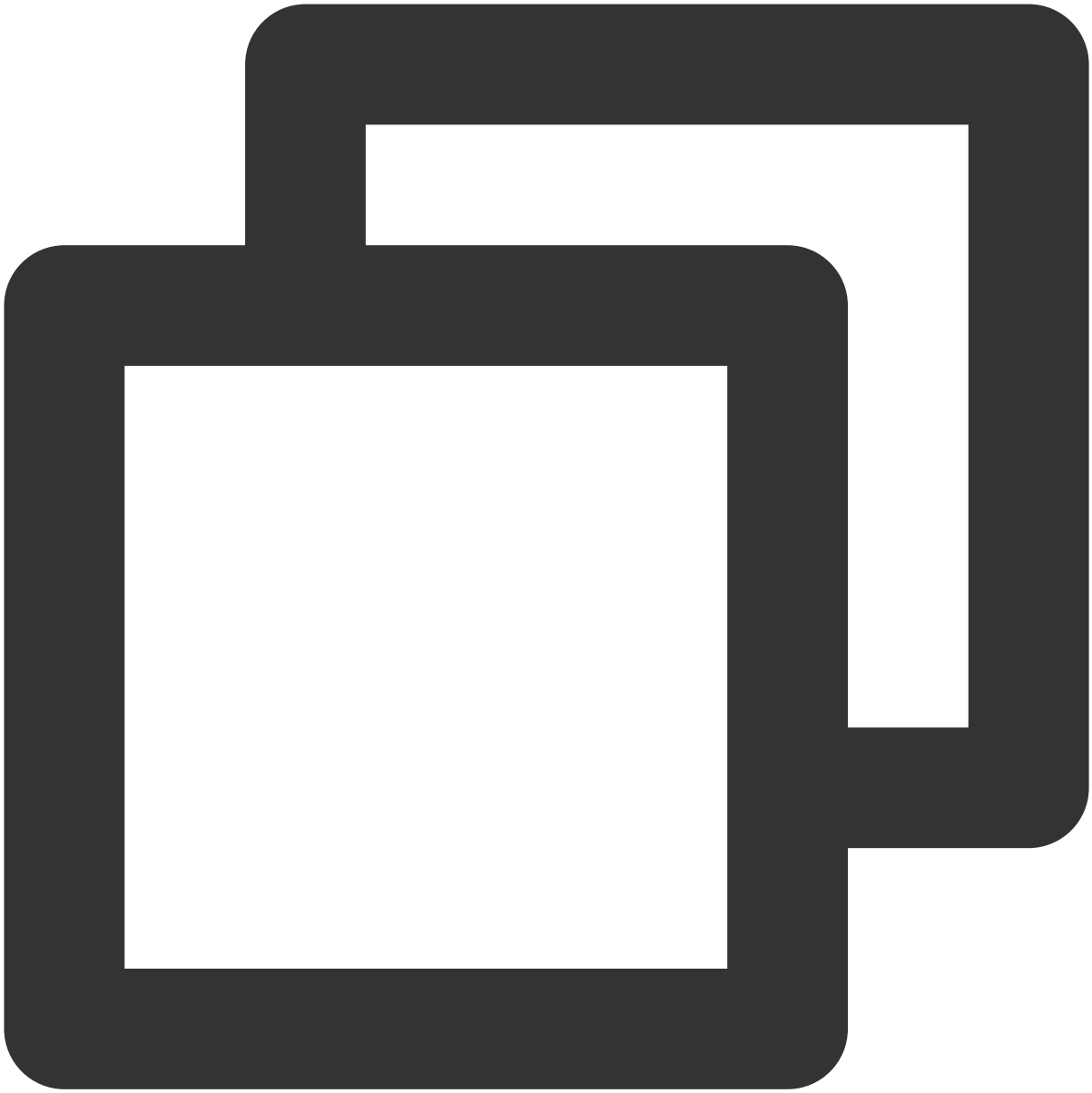
`giftName: String` : Gift Name

`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

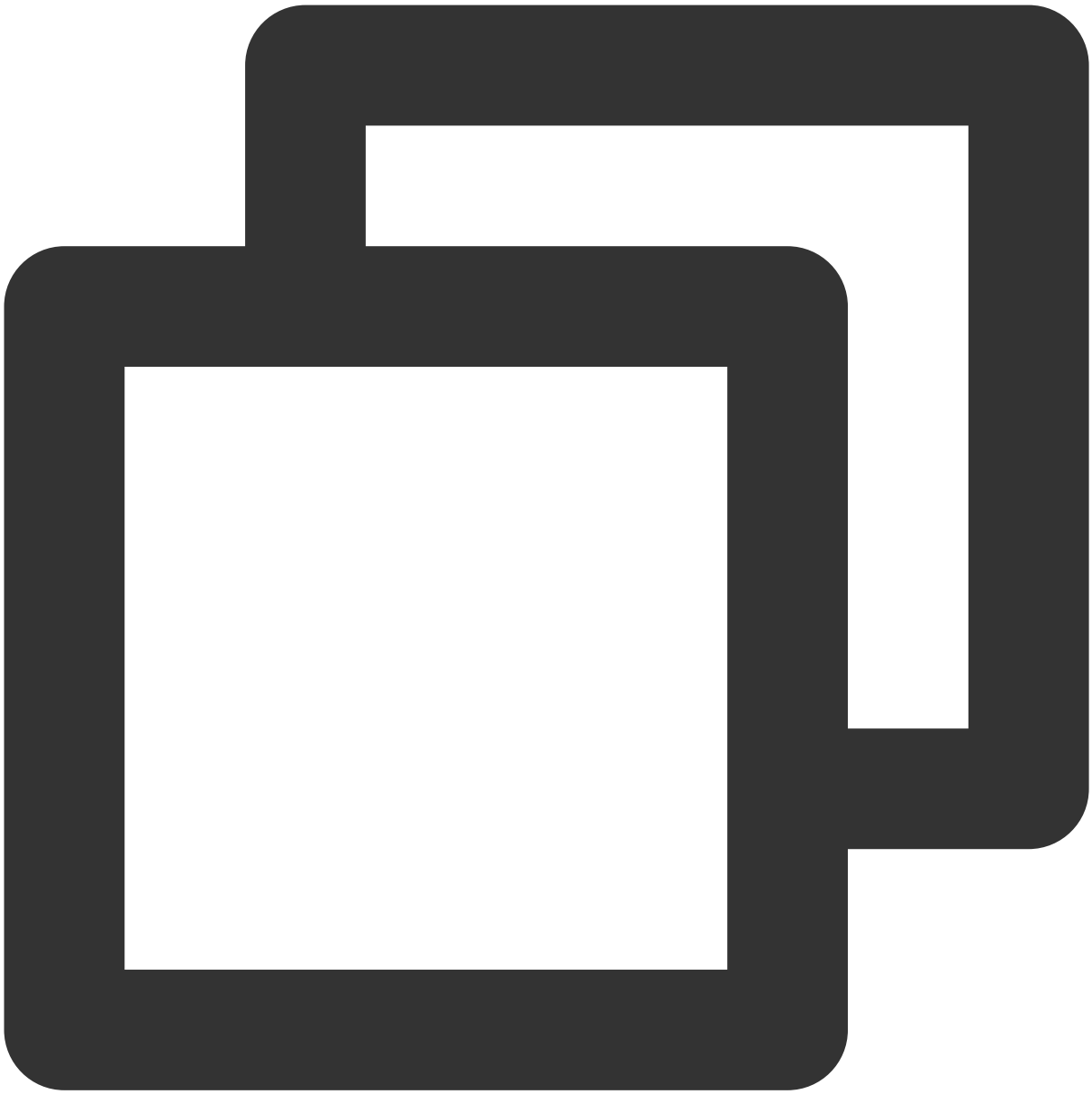
Implement the `onSendGift` callback in the `OnGiftListener` of `TUIGiftListView` , get the number of gifts and gift information, after preprocessing, you can call the `sendGift` function of `TUIGiftListView` for the actual sending of gifts.



```
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.

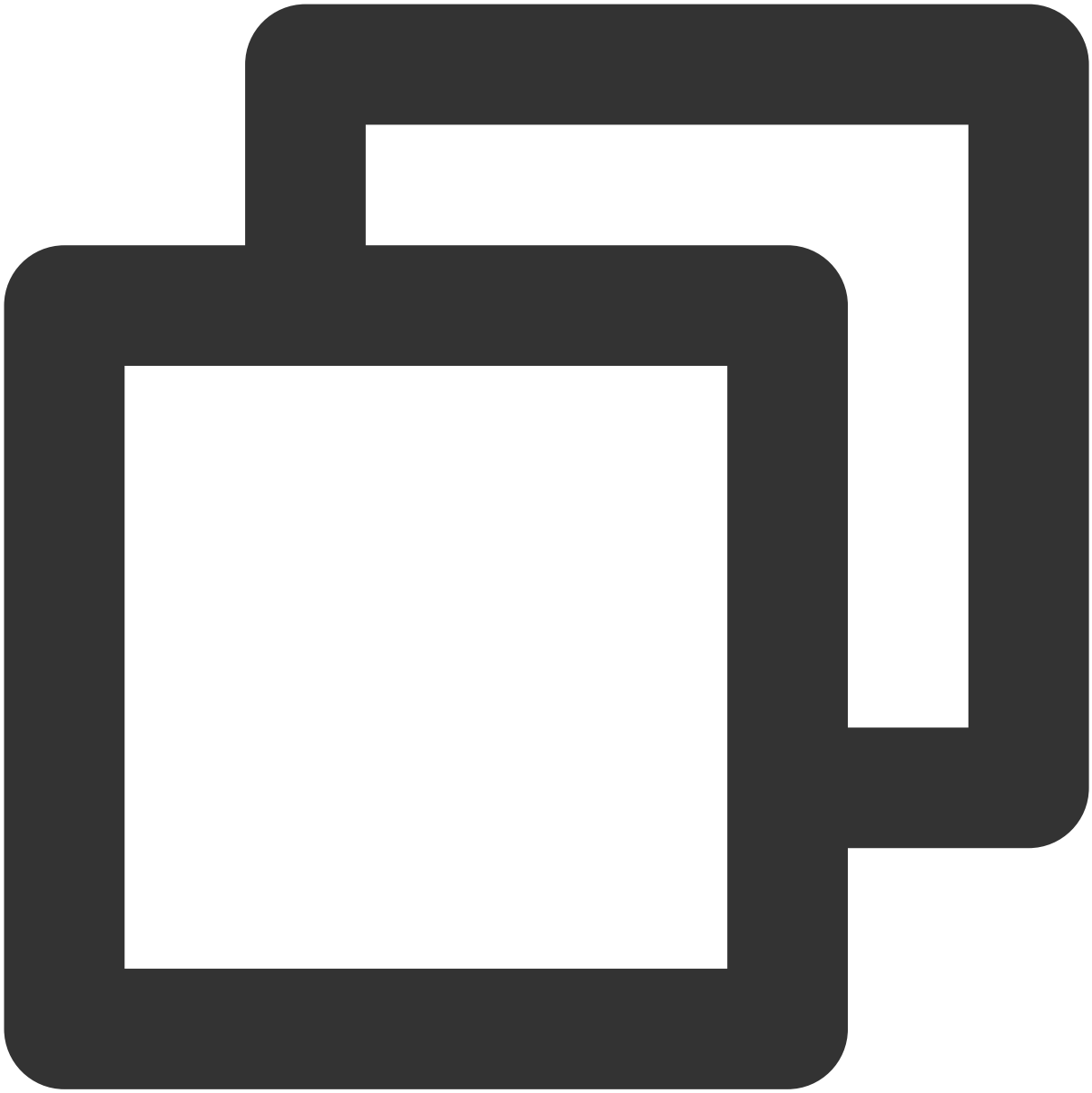


```
TUIGiftPlayView giftPlayView = new TUIGiftPlayView(mContext, roomId);
```

Note：

`TUIGiftPlayView` requires full-screen integration.

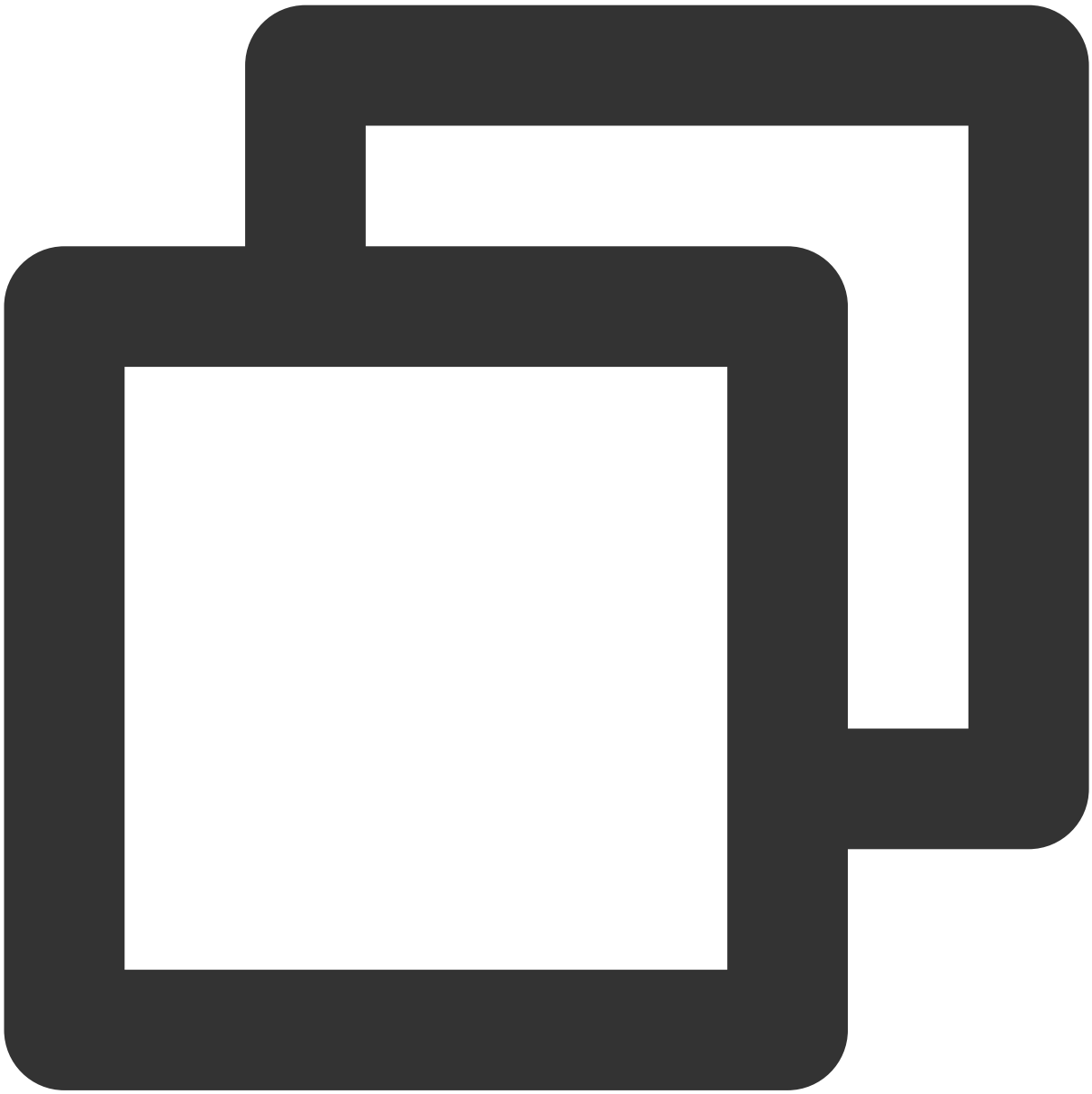
If you need to get the callback information of receiving gifts, you can implement the `onReceiveGift` callback in the `TUIGiftPlayViewListener` of `TUIGiftPlayView` .



```
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` .



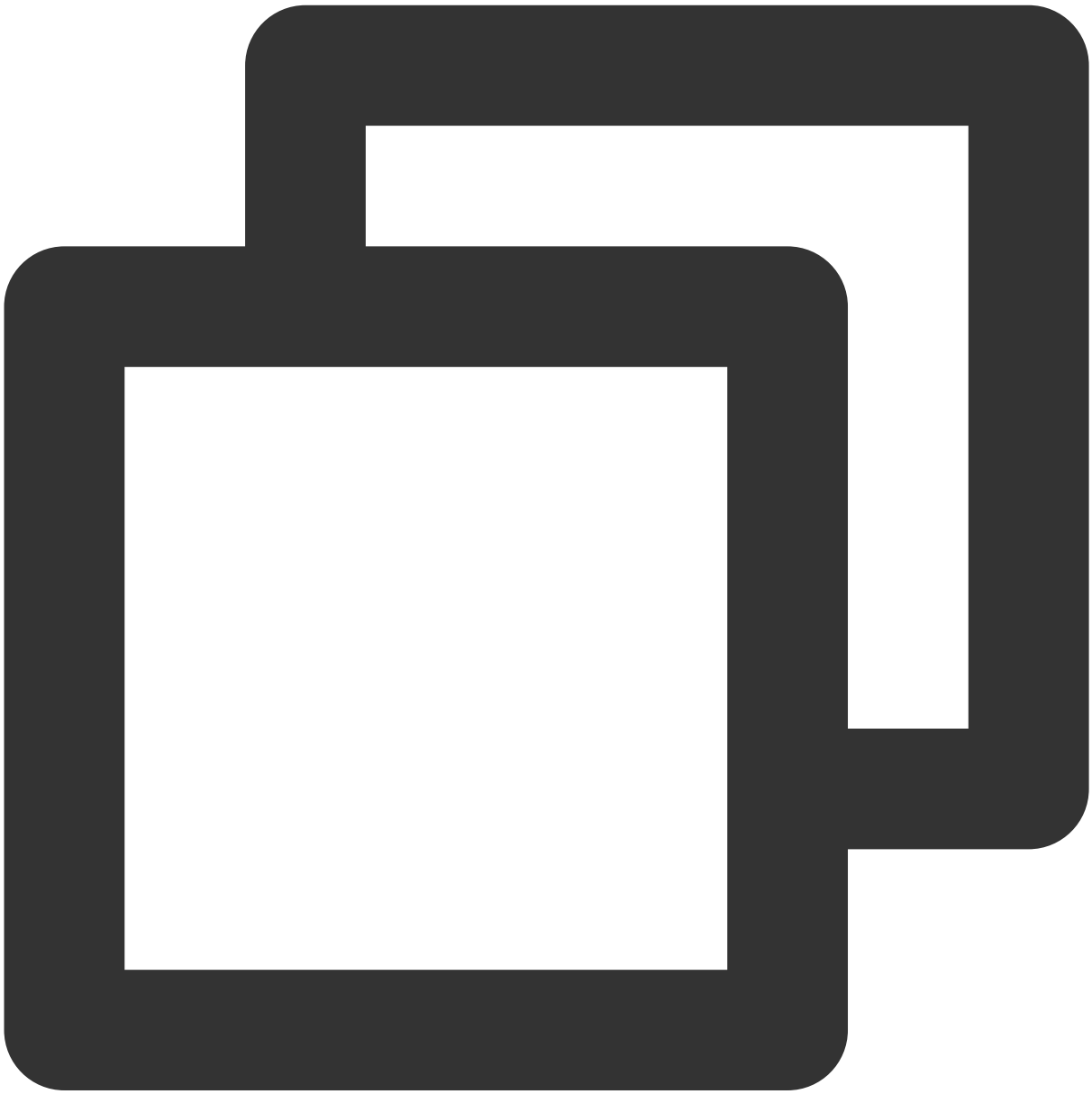
```
public interface TUIGiftPlayViewListener {
    void onPlayGiftAnimation(TUIGiftPlayView view, TUIGift gift);
    //...
}
```

Note：

Only SVGA animations are supported.

Set balance

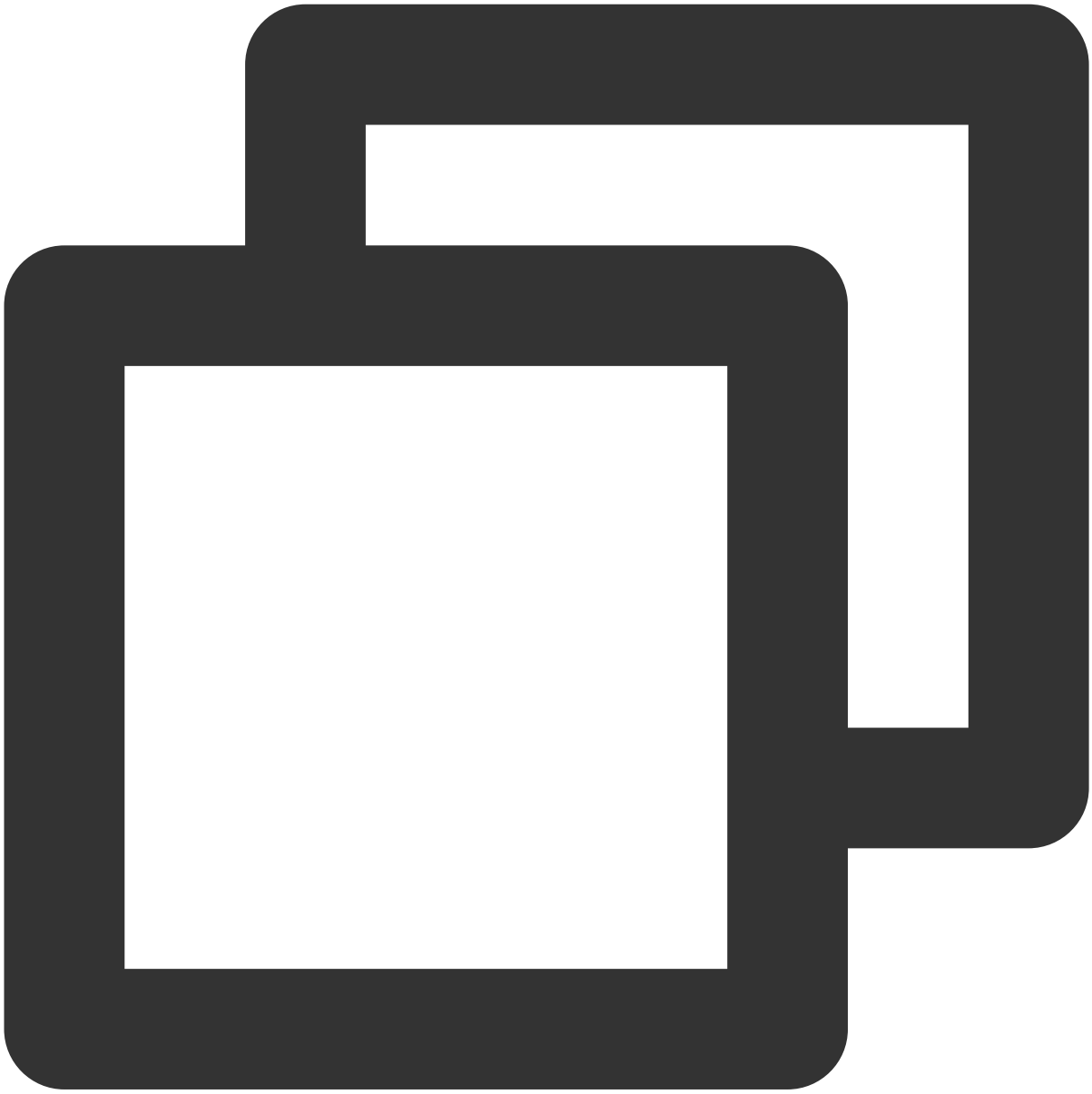
The gift panel component `TUIGiftListView` provides the `setBalance` interface, which can be used to set the balance value displayed on the gift panel.



```
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.



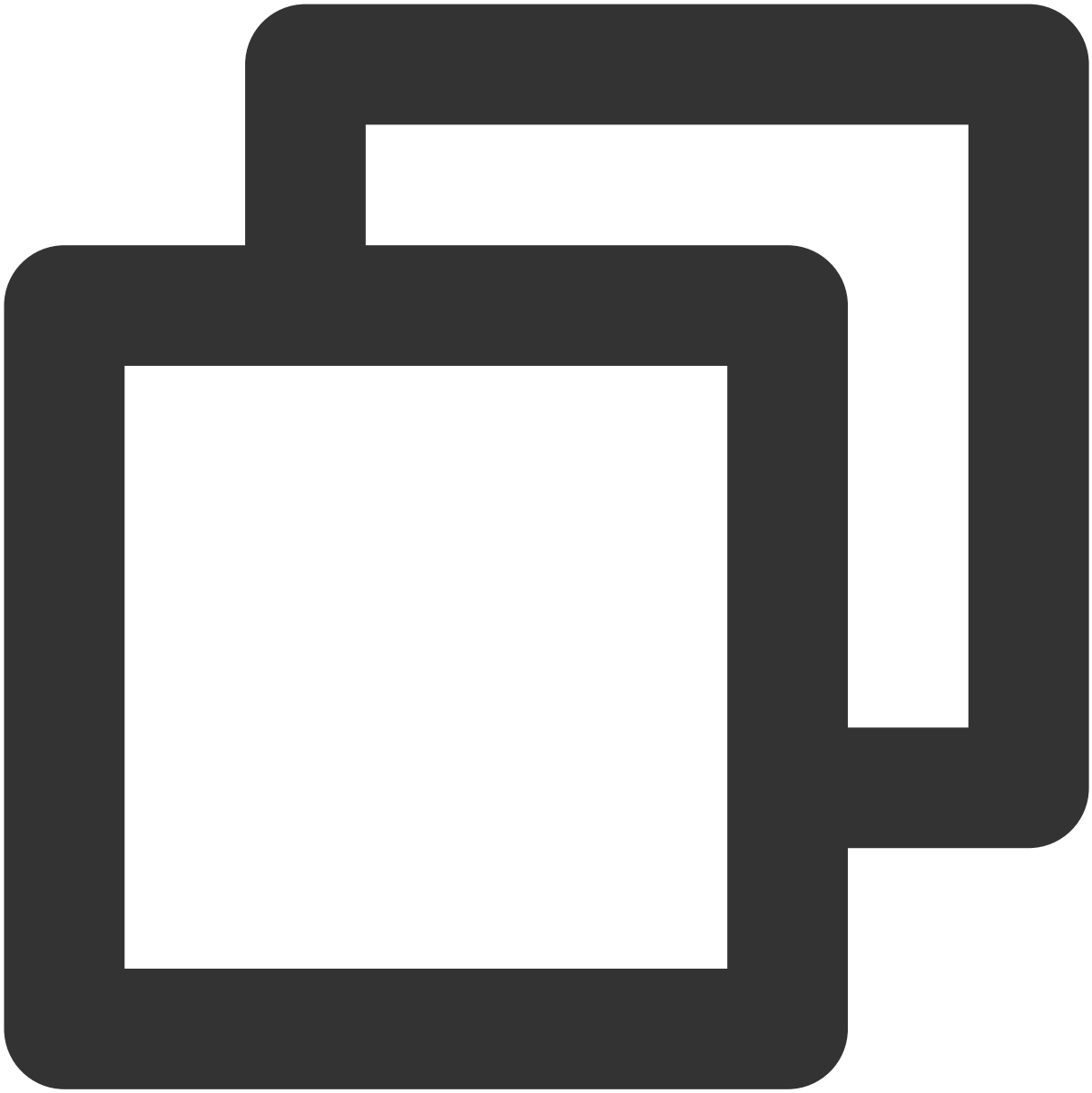
```
public void onRecharge(TUIGiftListView giftListView) {
    //...to recharge
    //setup the latest balance
    giftListView.setBalance(balance);
}
```

Note：

1. The gift balance is a concept of virtual currency, not real money.
2. 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

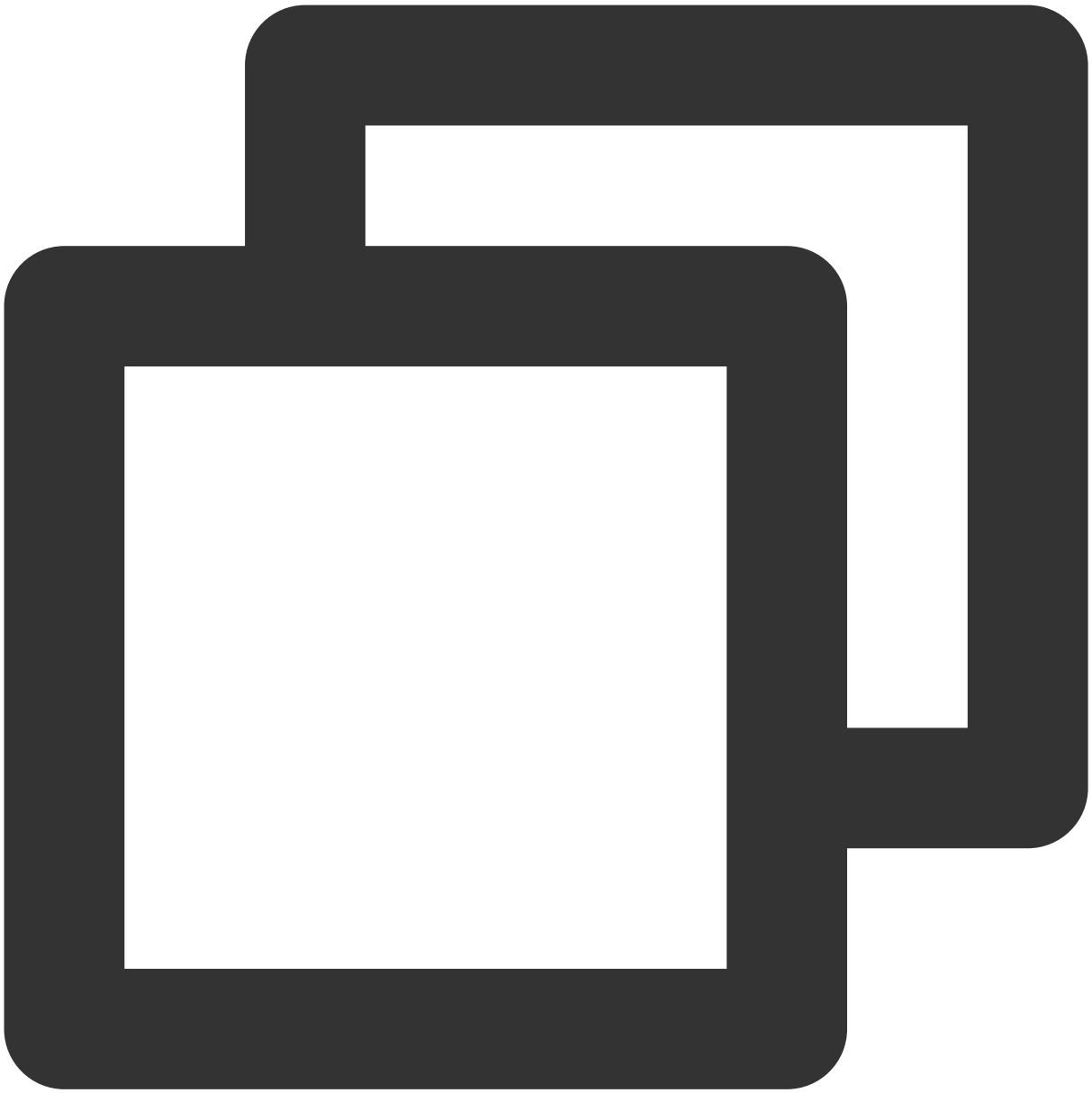
Implement the `onSendGift` callback in the `OnGiftListener` of `TUIGiftListView`, connect to the customer's own business server, complete the balance verification, gift billing, and consumption statistics, and then call the `sendGift` of `TUIGiftListView` to send the gift message.



```
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:

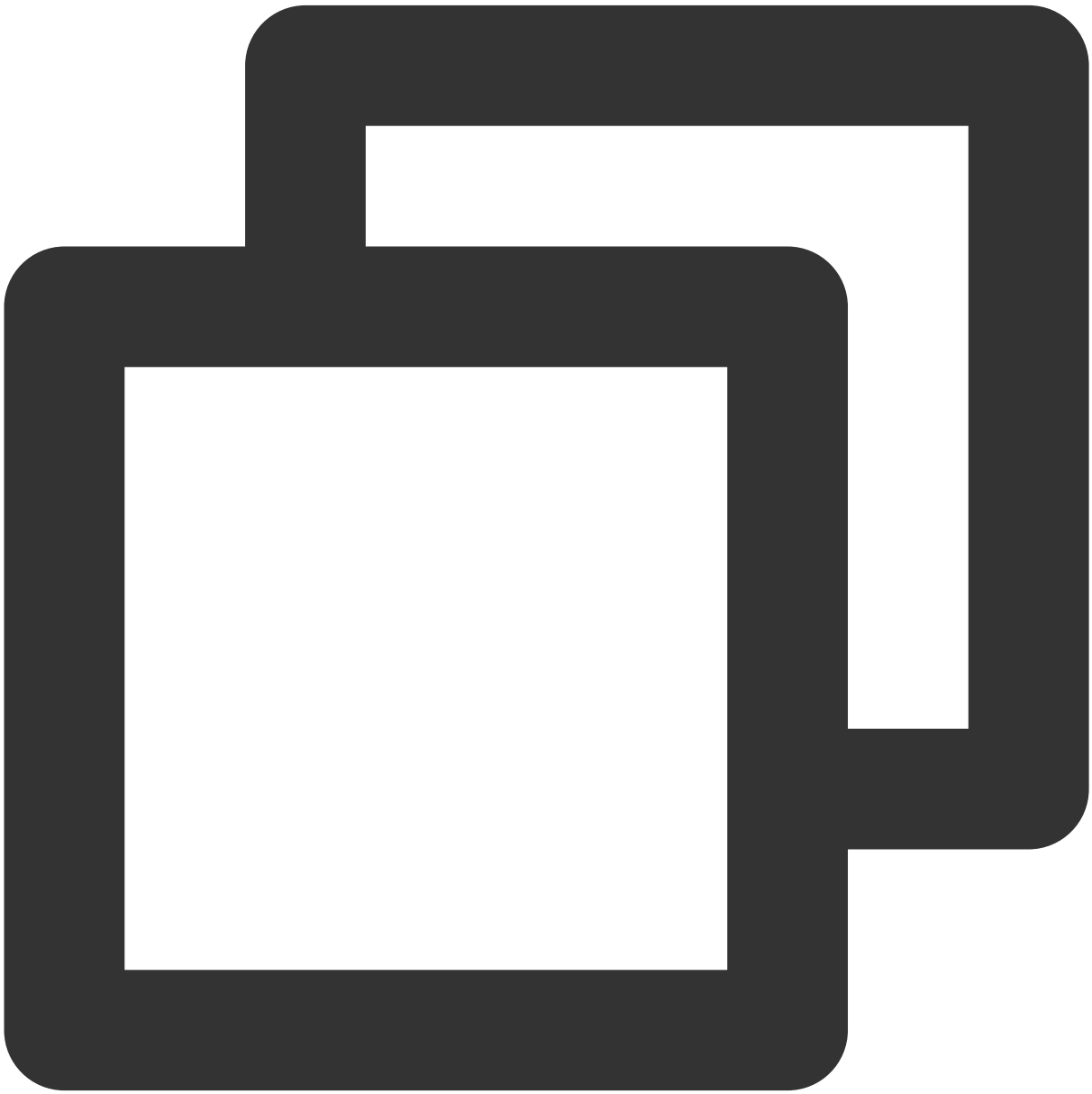


```
// Source code path：tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/a

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

Note：

1. 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`.
2. The `animationUrl` of the gift is required to be a SVGA animation.

Customize giftPanel`s balance



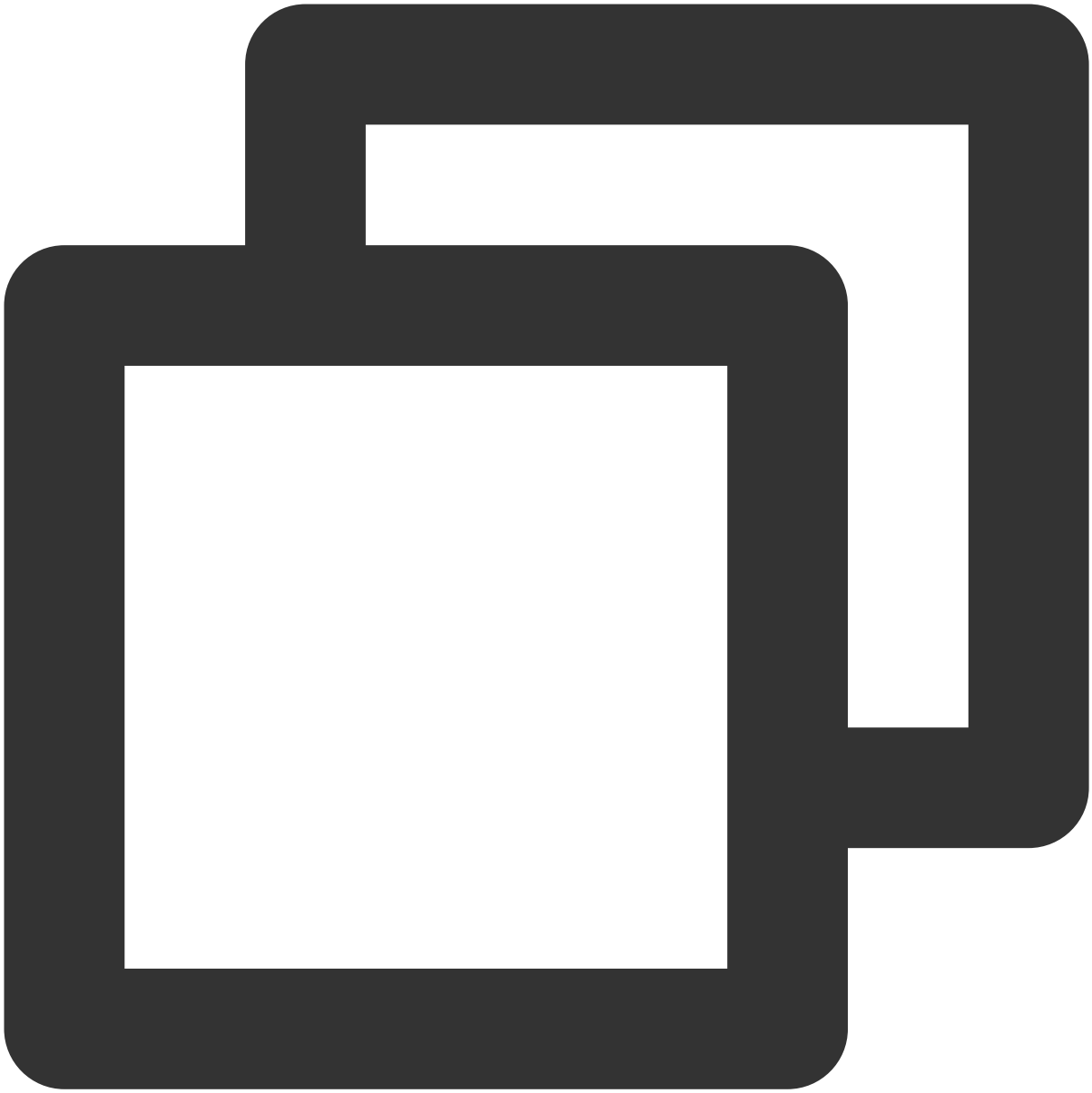
```
// Source code path：tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/a
```

```
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



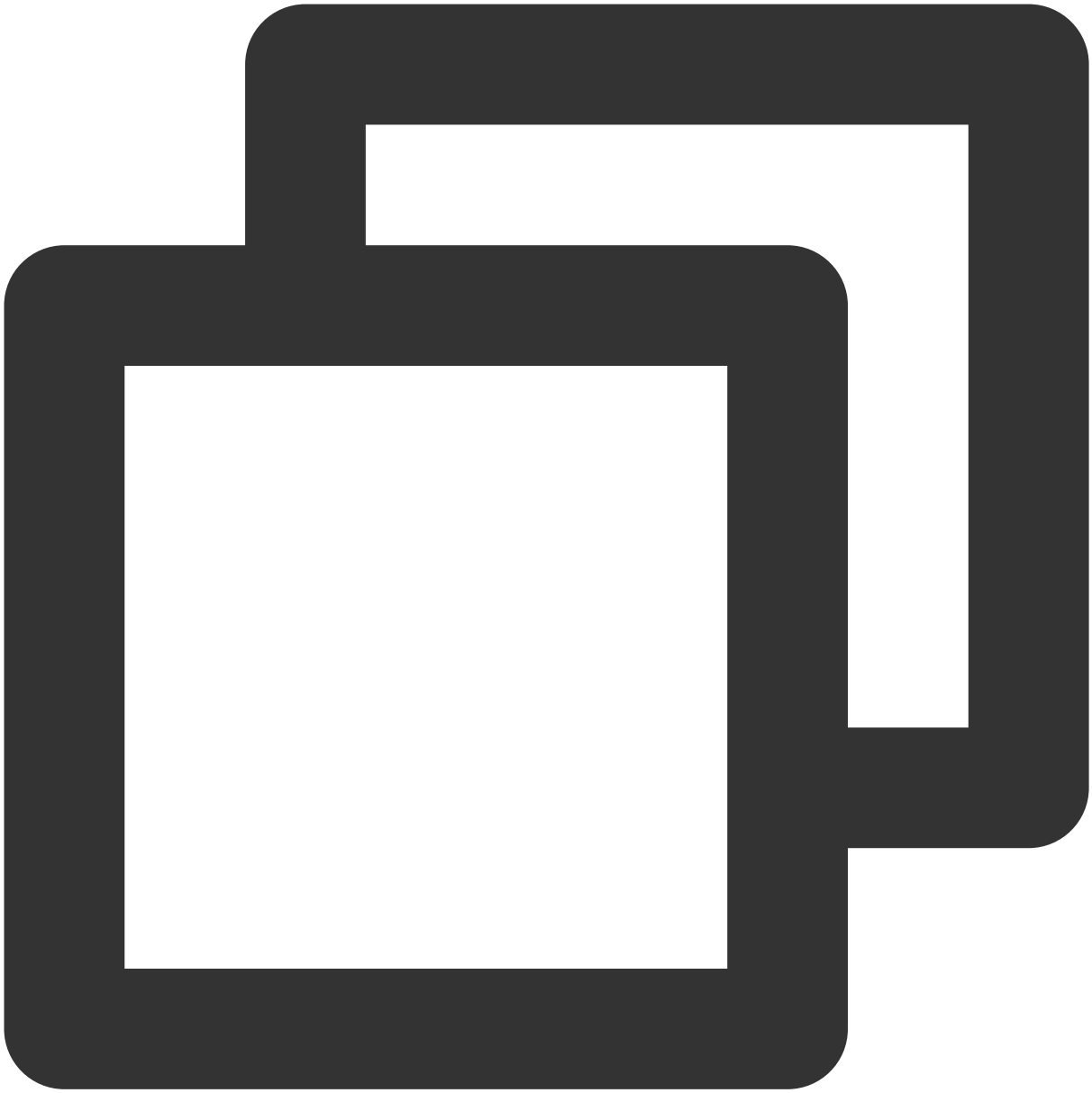
```
// Source code path：tuilivekit/src/main/java/com/trtc/uikit/livekit/liveroom/view/a

@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` .

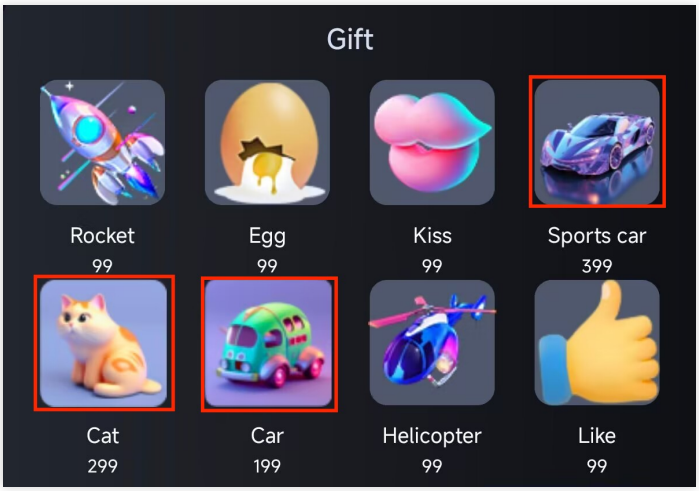
Gift Effects (TUILiveKit)
Android

Last updated：2024-08-02 16:51:31

TUILiveKit provides two types of gift effect players: the basic effect player and the advanced effect player. By default, the basic effect player is integrated. If you have higher performance requirements for the player or expect support for more animation file formats, we also provide an advanced effect player for your use.

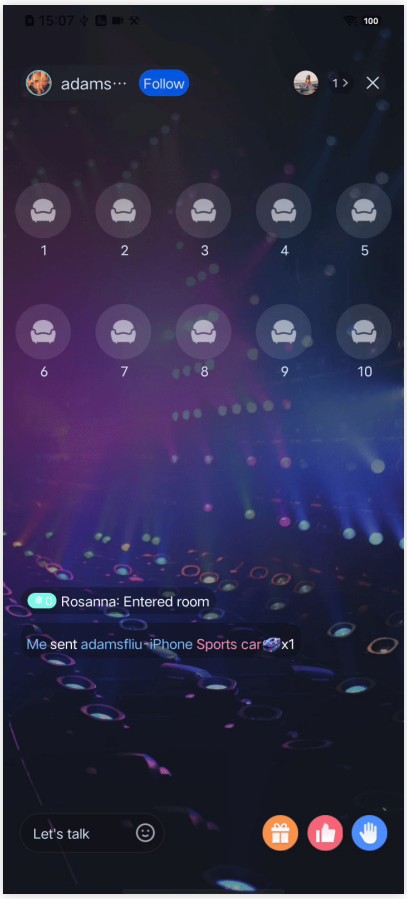
Basic effect player

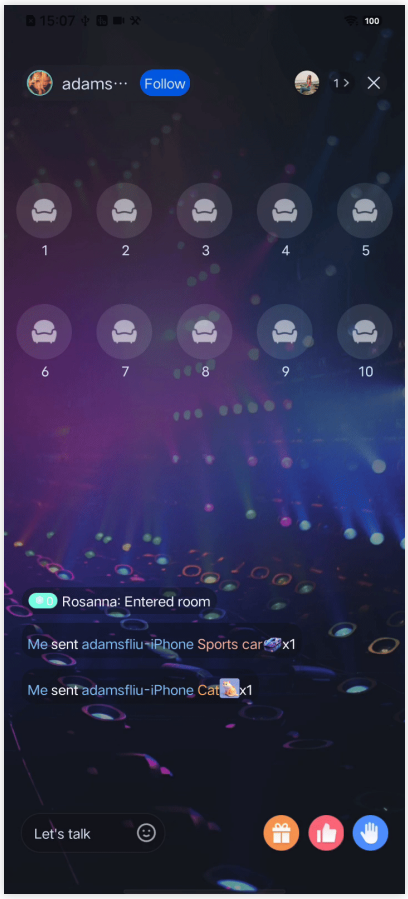
The basic effect player is based on `SVGA` and supports only `SVGA` format files for special effect animations. When using the basic effect player, it comes with the following three default special effect animations:

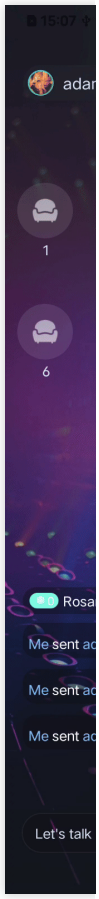


Effect Showcase

Sports car	Cat	Car



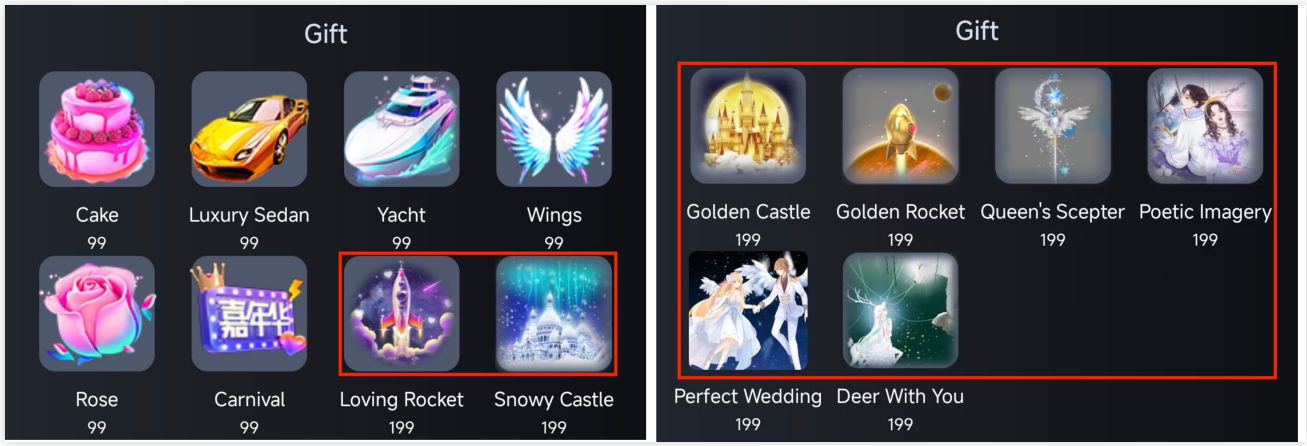




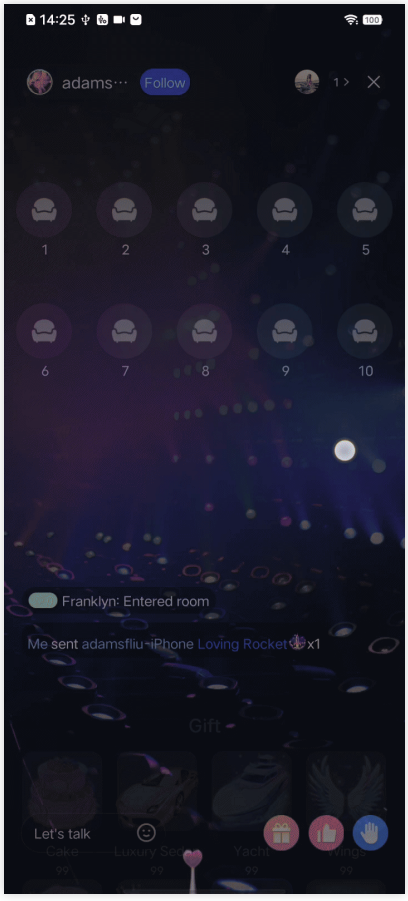
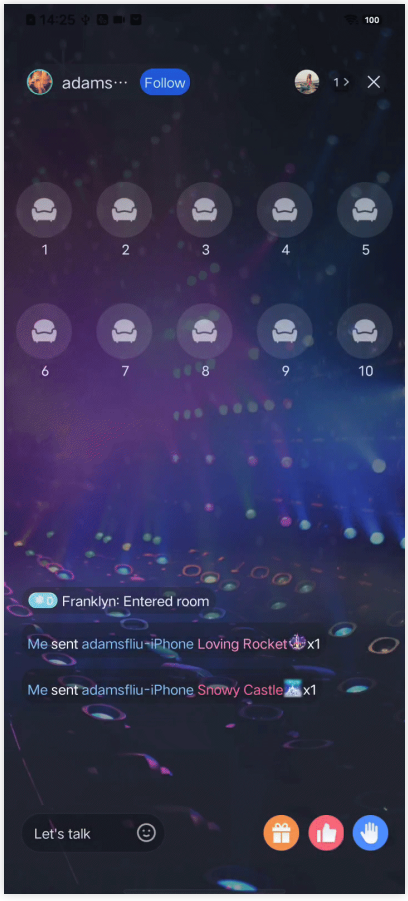
Advanced Effect Player

The TUILiveKit advanced effect player adopts the **Tencent Effect Player** and supports various formats of special effect animations.The advanced effect player supports various formats of effect animations, such as vap, Lottie, mp4, svga, and more.

When using the advanced effect player, it comes with the following eight default special effect animations:



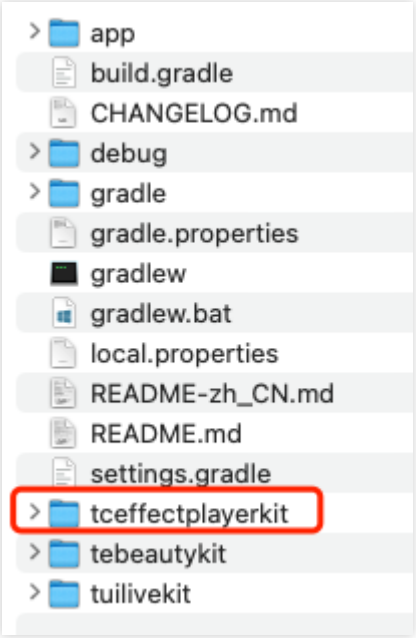
Effect Showcase

Loving Rocket	Snowy Castle	Deer With Yo
		

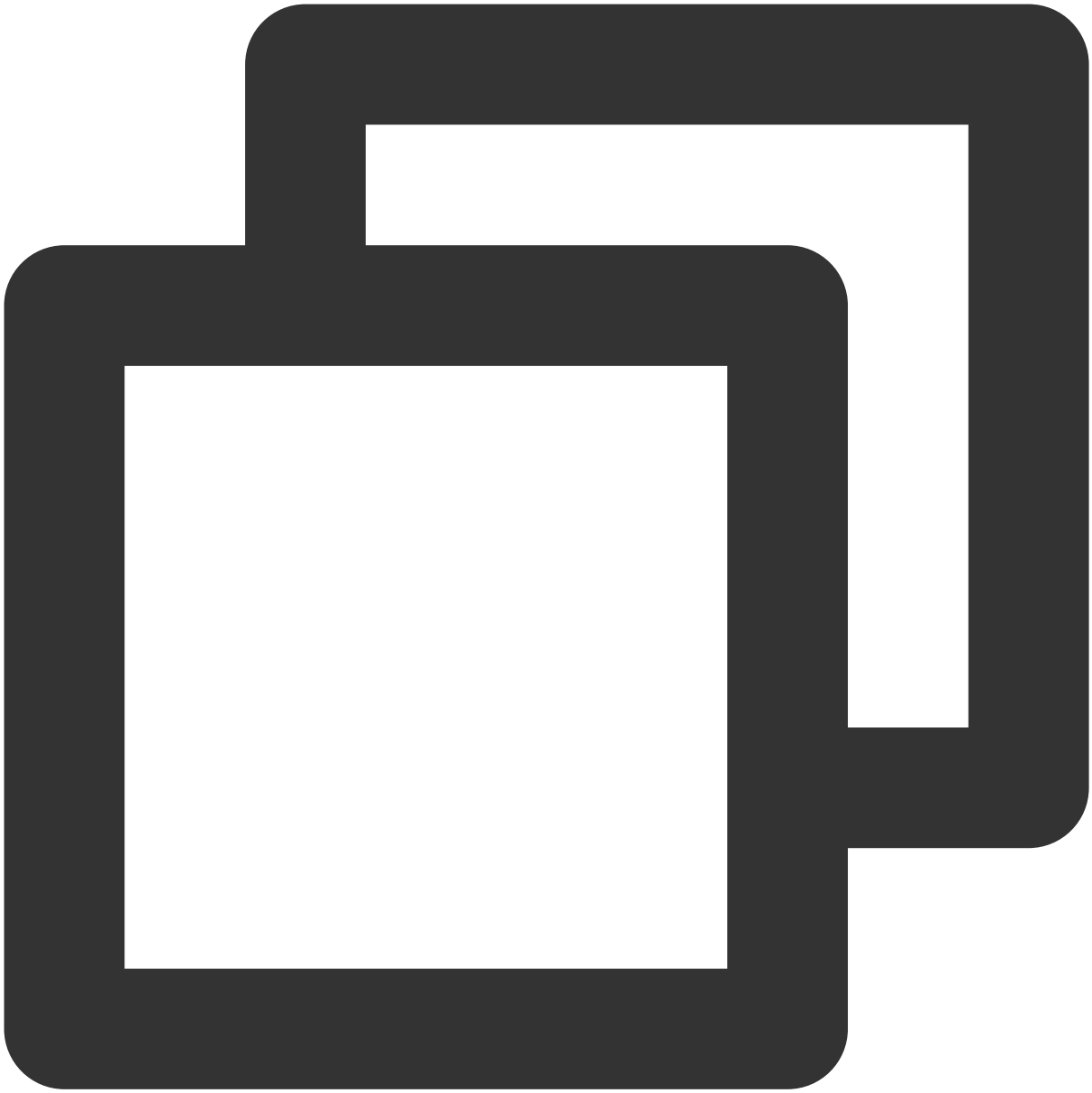
Integration Guide

Step 1: Integrating `tceffectplayerkit`

1. Download and extract TUILiveKit. Copy the `Android/tceffectplayerkit` folder to your project, at the same level as the app folder.



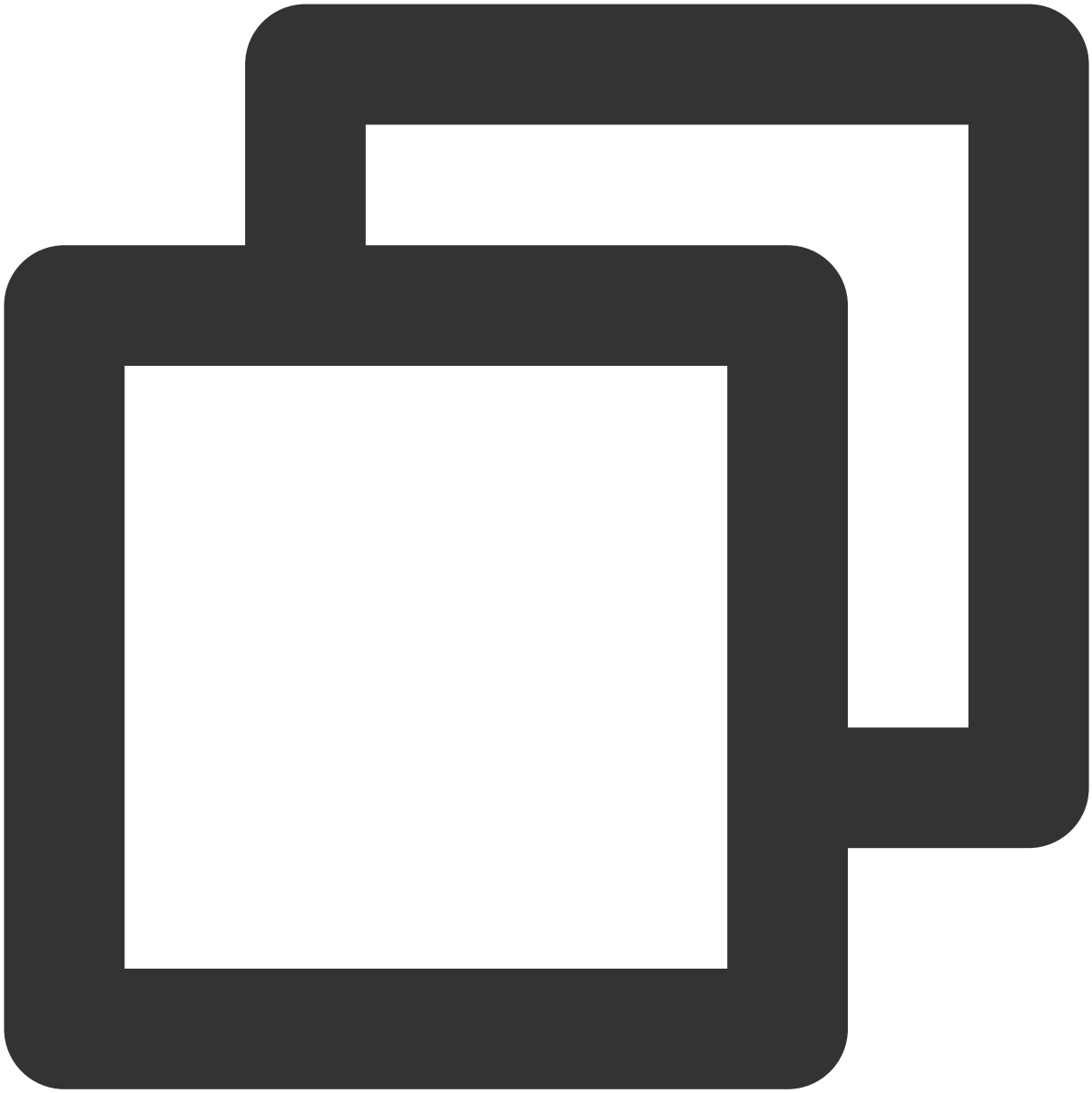
2. Please edit your project's `settings.gradle` file and add the following code:



```
include ':tceffectplayerkit'
```

Step 2: Authorization

1. Apply for authorization and obtain `LicenseUrl` and `LicenseKey`, please contact TRTC_helper@tencent.com for more details.
2. In the initialization section of your business logic (typically in the same location as the login process), add the following authorization code and replace it with the `LicenseUrl` and `LicenseKey` you have obtained:



```
TCMediaXBase.getInstance().setLicense(context,
      "LicenseUrl", // Replace with your LicenseUrl
      "LicenseKey", // Replace with your LicenseKey
      new ILicenseCallback() {
          @Override
          public void onResult(int error, String message) {
              Log.i("TCMediaXBase", "setLicense result: " + error + " " + message
          }
      });
```

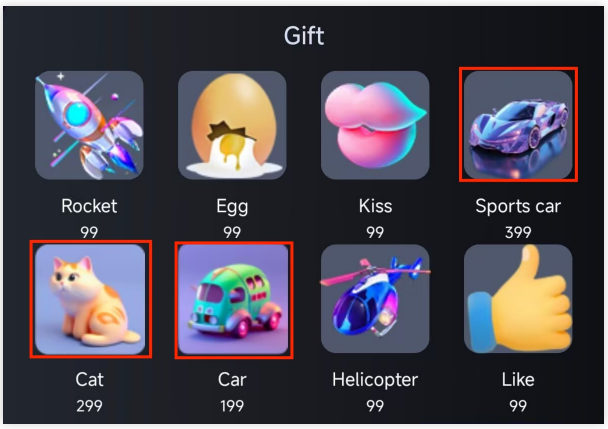
iOS

Last updated：2024-08-02 16:53:11

TUILiveKit provides two types of gift effect players: the basic effect player and the advanced effect player. By default, the basic effect player is integrated. If you have higher performance requirements for the player or expect support for more animation file formats, we also provide an advanced effect player for your use.

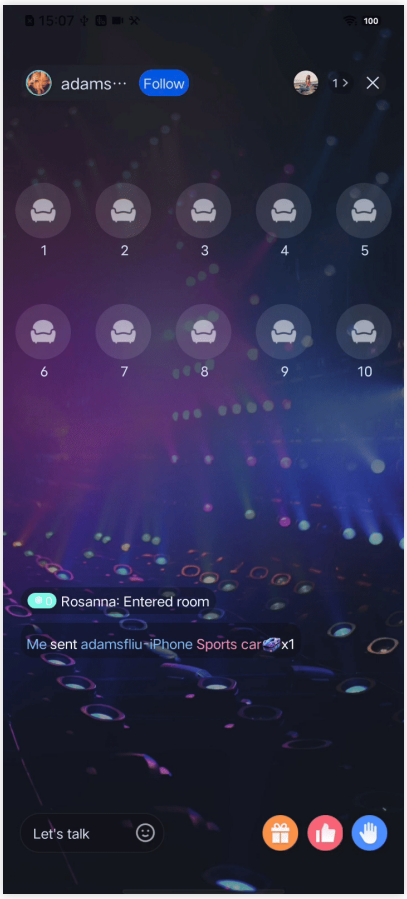
Basic Effect Player

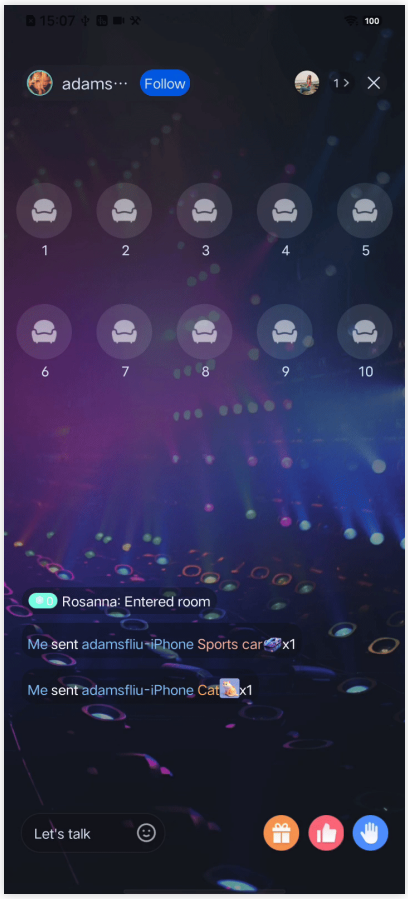
The basic effect player is based on `SVGA` and supports only `SVGA` format files for special effect animations. When using the basic effect player, it comes with the following three default special effect animations:

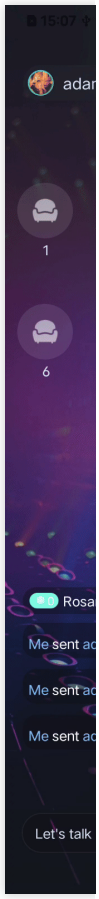


Effect Showcase

Sports car	Cat	Car



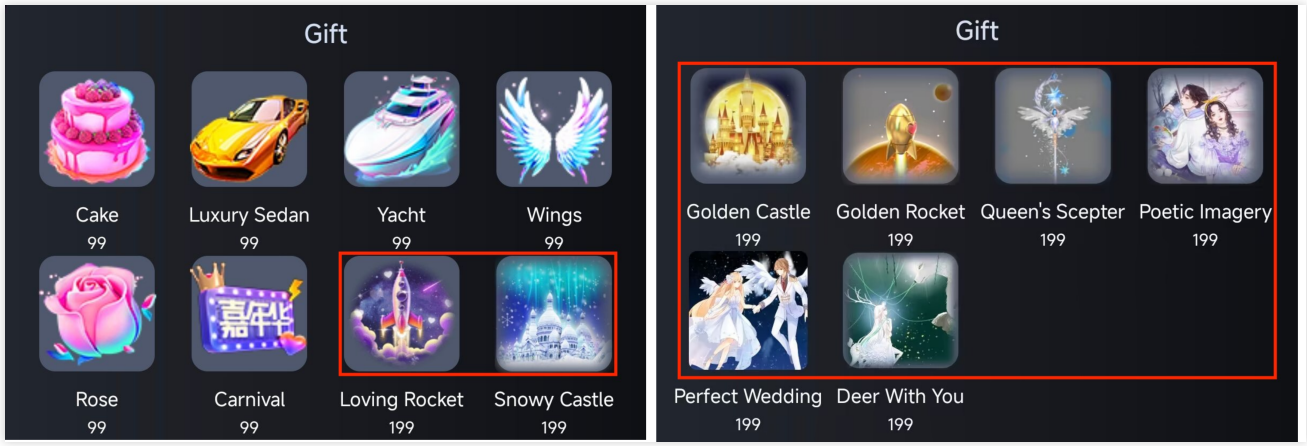




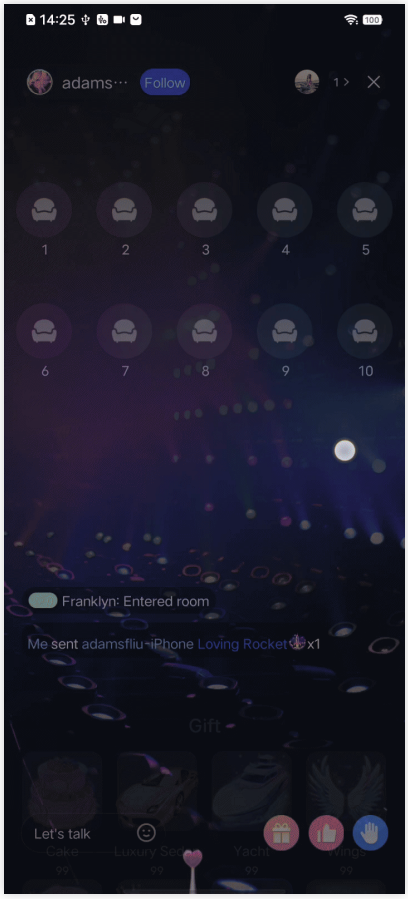
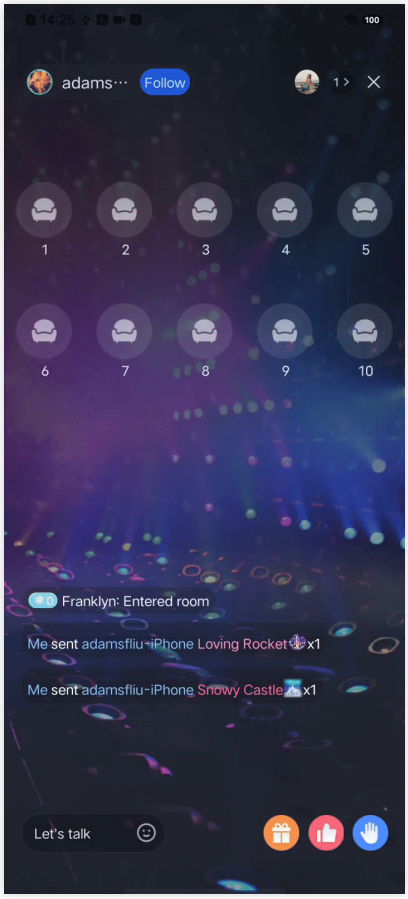
Advanced Effect Player

The TUILiveKit advanced effect player adopts the **Tencent Effect Player** and supports various formats of special effect animations. The advanced effect player supports various formats of effect animations, such as vap, Lottie, mp4, svga, and more.

When using the advanced effect player, it comes with the following eight default special effect animations:



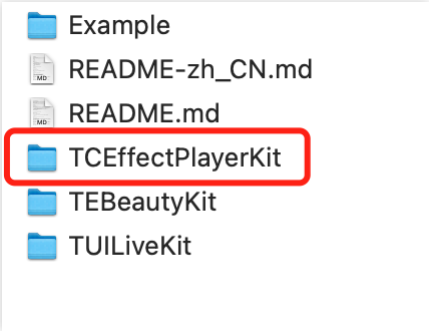
Effect Showcase

Loving Rocket	Snowy Castle	Deer With Yo
		

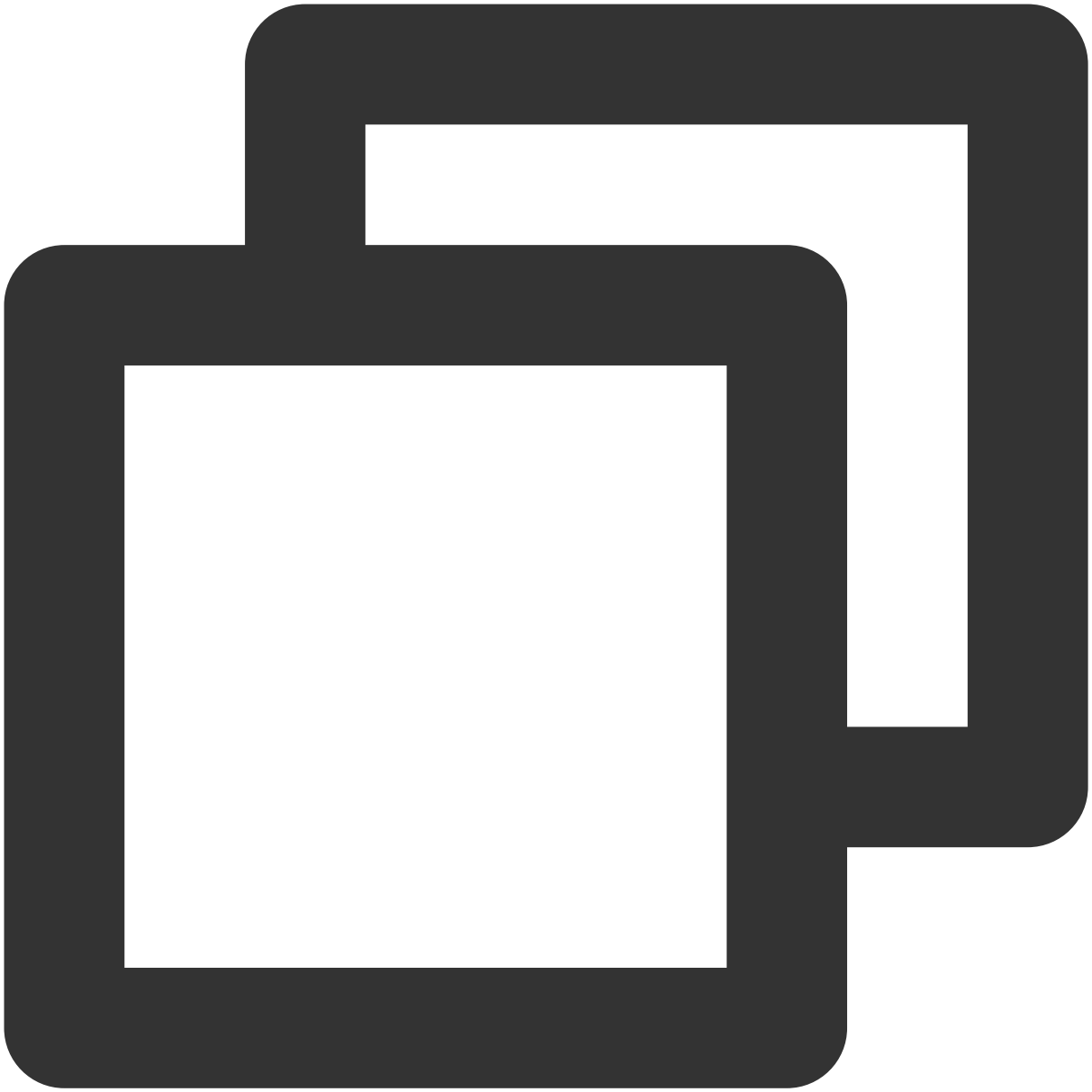
Integration Guide

Step 1: Integrating `TCEffectPlayerKit`

1. Download and extract TUILiveKit. Copy the `iOS/TCEffectPlayerKit` folder to your project, at the same level as the `Podfile` folder.



2. Please edit the `Podfile` and add the following code:



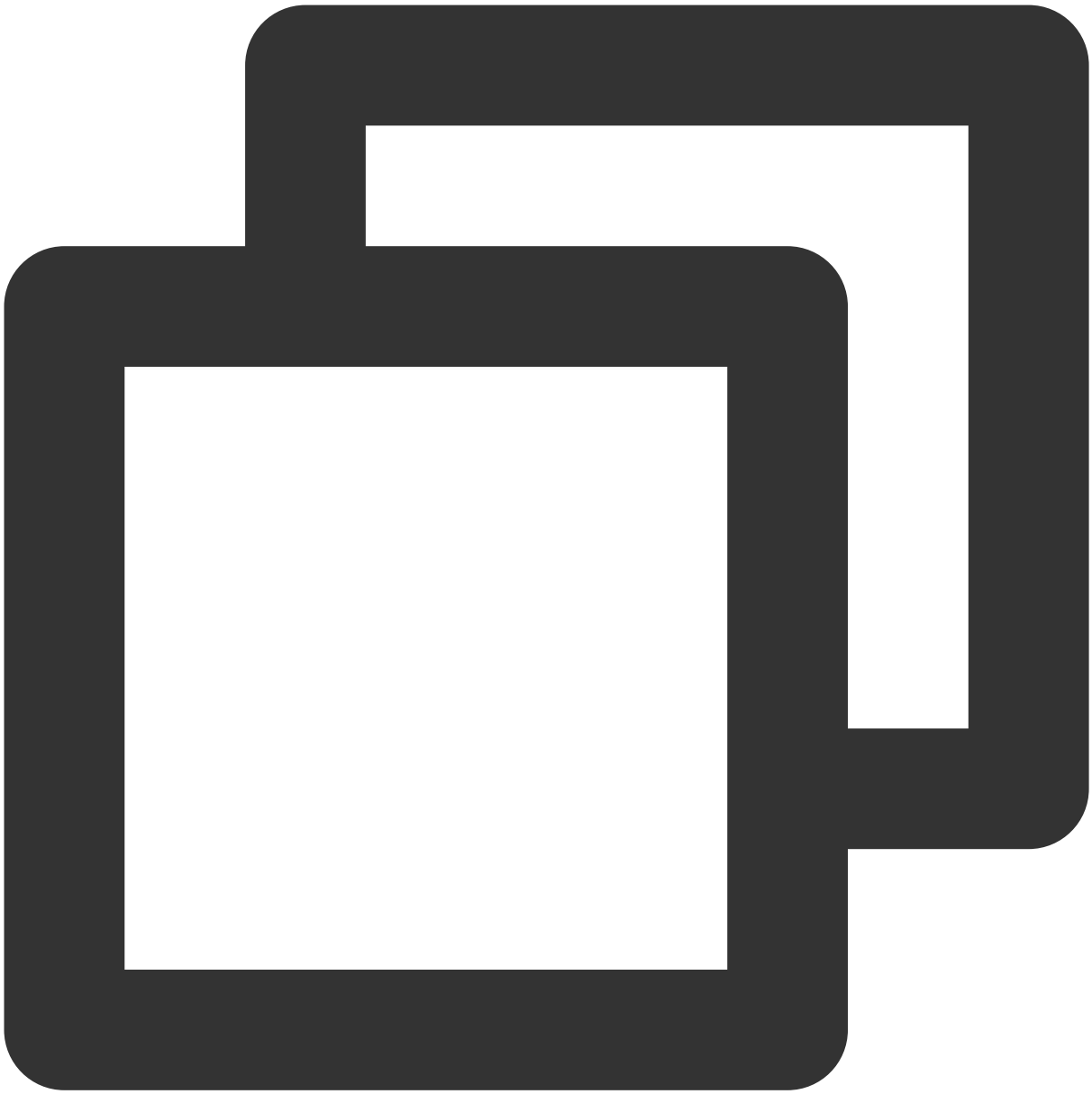
```
pod 'TCEffectPlayerKit',:podspec => './TCEffectPlayerKit/TCEffectPlayerKit.pods
```

3. Save the changes and run `pod install` in the terminal to install the TCEffectPlayerKit dependency.

Step 2: Authorization

1. Apply for authorization and obtain `LicenseURL` and LicenseKEY, please contact TRTC_helper@tencent.com for more details.
2. Set the `URL` and `KEY` in the initialization code of the relevant business module, and configure the resources for the special effect player. For example, in iOS, you can set these in the `didFinishLaunchingWithOptions`

method of the `AppDelegate` .



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

import TCMediaX

func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions: [UIApplication.Launch
        TCMediaXBase.getInstance().setLicenceURL("LicenseURL", key: "LicenseKEY")
```

```
        return true
}
```

Beauty Effects (TUILiveKit)
Android

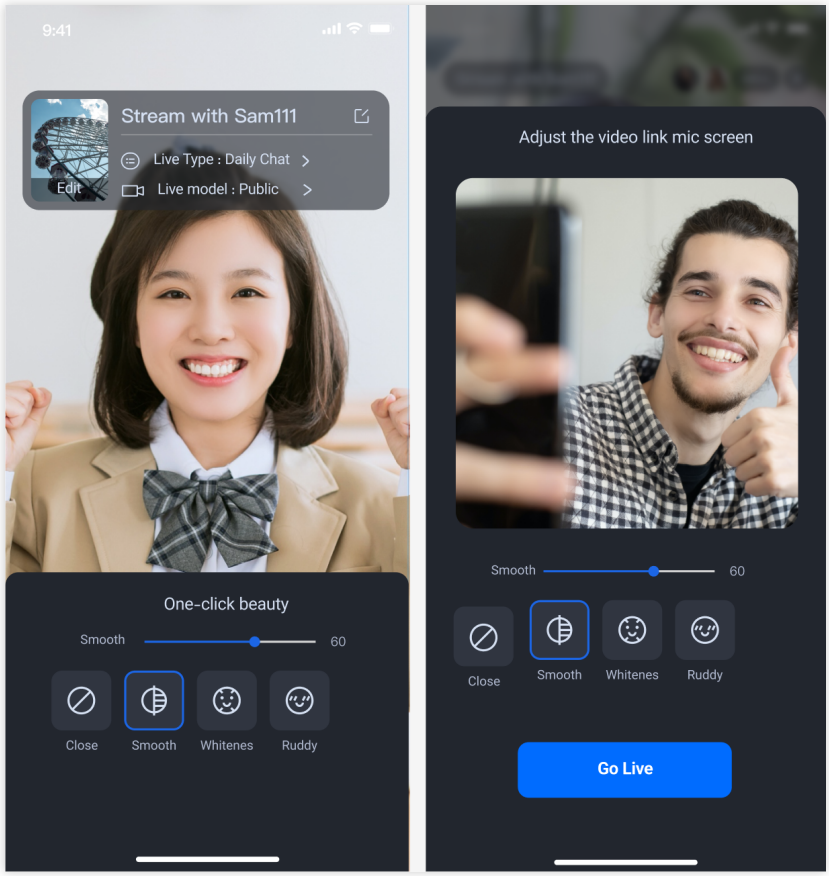
Last updated：2024-08-02 16:36:24

TUILiveKit offers two types of beauty effects: `Basic Beauty` and `Advanced Beauty`. If you are not satisfied with the results of Basic Beauty, you can choose to integrate Advanced Beauty to meet your more advanced beauty needs.

Basic Beauty

TUILiveKit comes with Basic Beauty functionality by default. Basic Beauty includes features such as skin whitening, skin smoothing, and adding a ruddy tint to the complexion. You can adjust the intensity of these beauty effects to meet different requirements. These features are already built-in within TUILiveKit, **so there is no need for additional configuration or integration.**

Panel Display



Advanced Beauty

TUILiveKit utilizes Tencent Effects Beauty for advanced beauty effects.

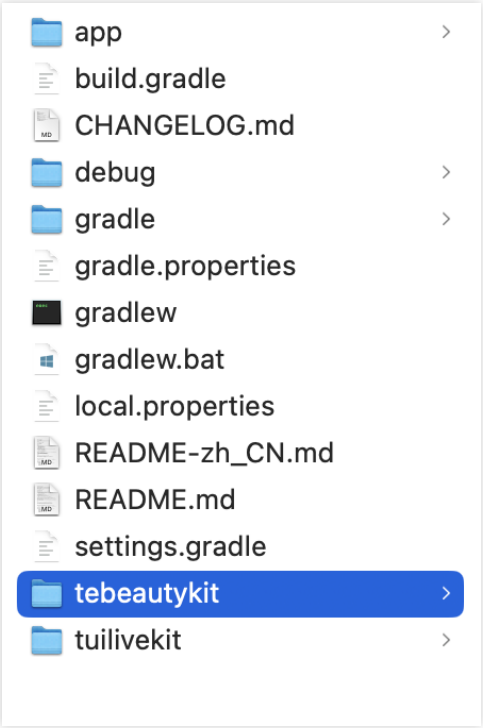
Effect Showcase

Chin slimming	Eye distance	Nose slimming

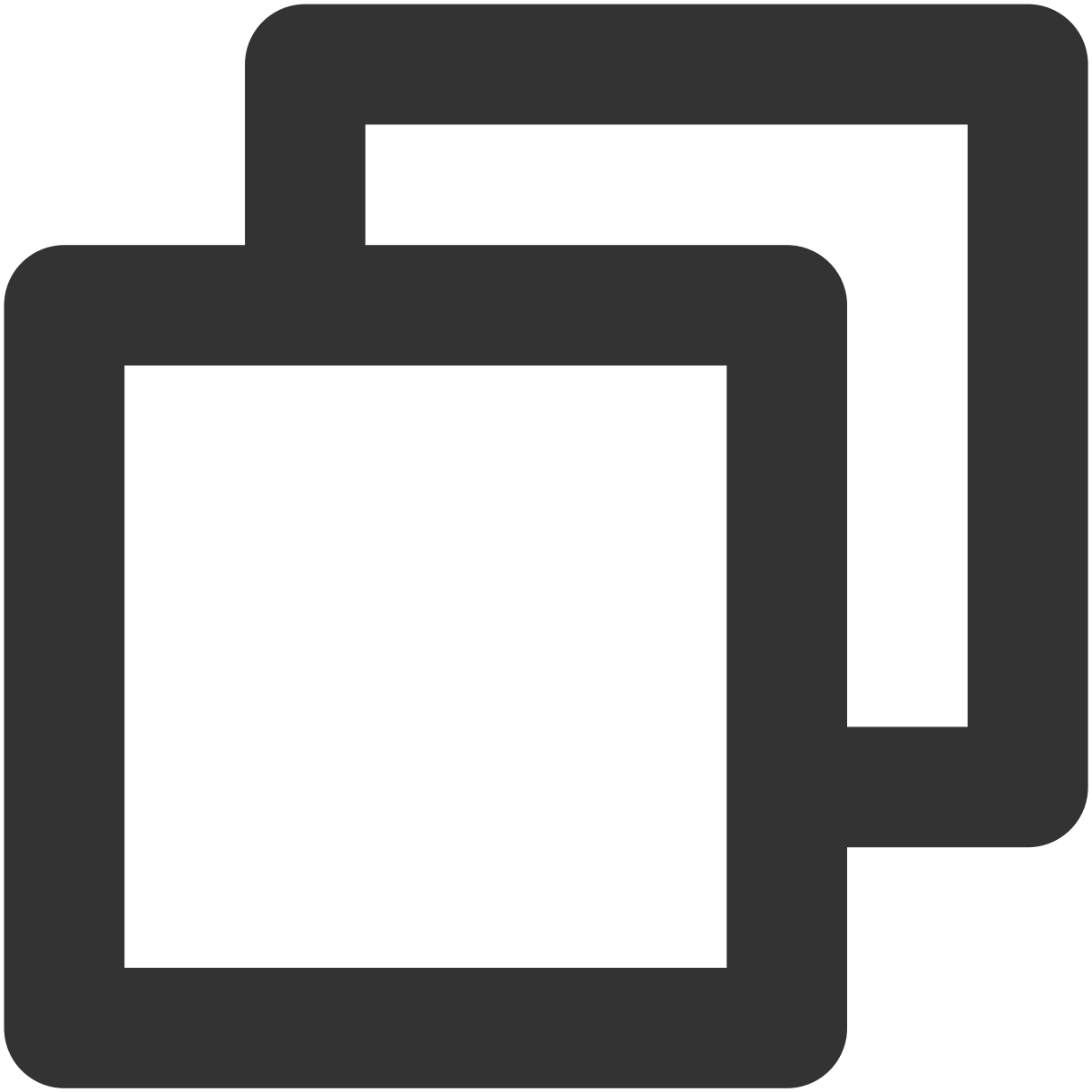
Integration Guide

Step 1: Integrating `tebeautykit`

1. Download and extract TUILiveKit. Copy the `Android/tebeautykit` folder to your project, at the same level as the app folder.



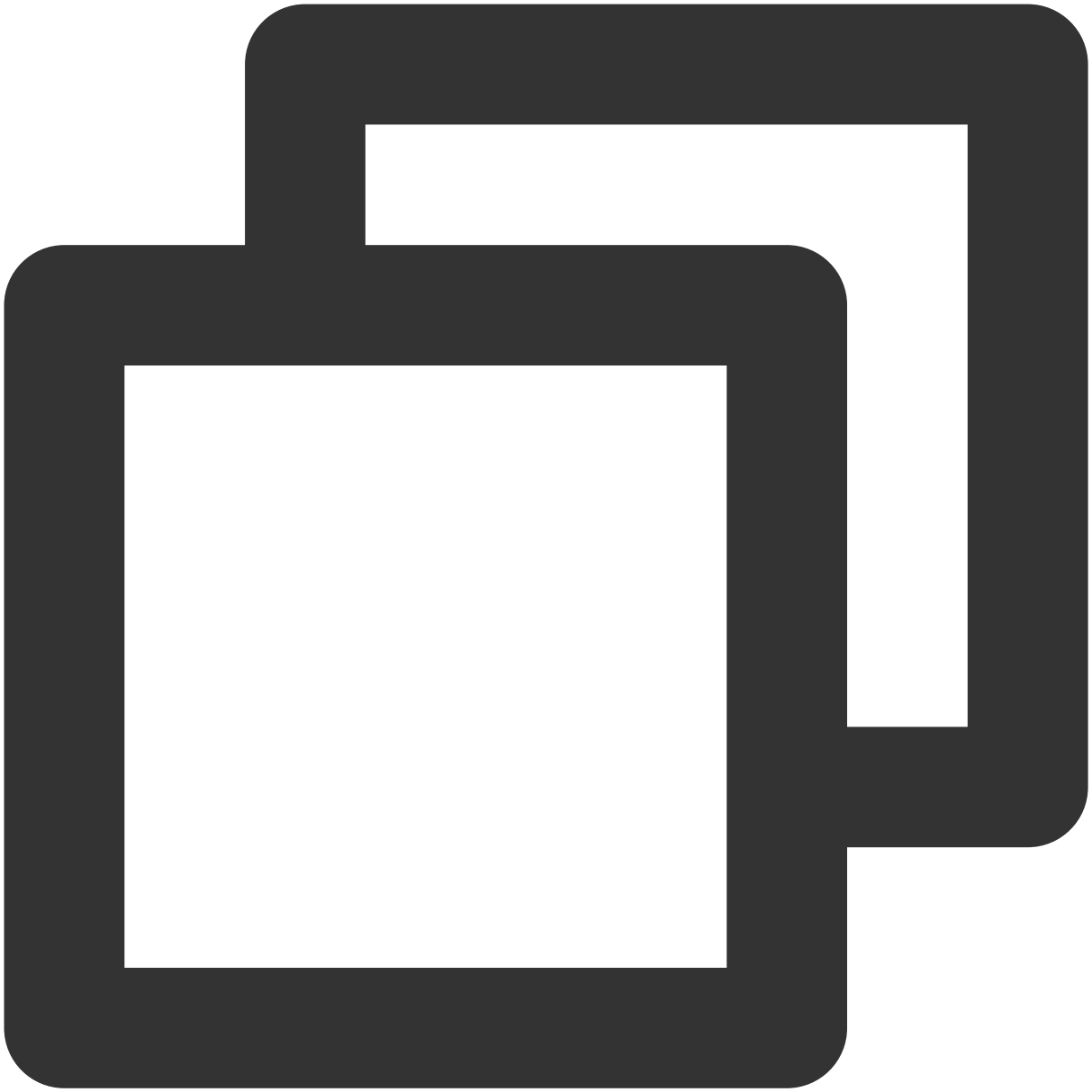
2. Please edit your project's `settings.gradle` file and add the following code:



```
include ':tebeautykit'
```

Step 2: Authorization & Setting Beauty Resources

1. Apply for authorization and obtain `LicenseUrl` and `LicenseKey`. Please refer to the License Guide for more information.
2. In the initialization section of your business logic (typically in the same location as the login process), Add the following authorization code and replace it with the `Beauty Package ID`, `LicenseUrl`, and `LicenseKey` you have obtained:



```
    TEBeautySettings.getInstance().initBeautySettings(context,
                                        S1_07,         // Replace S1_07 with the bea
                                        "LicenseUrl",  // Replace with your LicenseU
                                        "LicenseKey"); // Replace with youLicenseKey
```

Note:

If you are unsure about the beauty package number, click here to view the overview of beauty package numbers.

By completing the aforementioned steps, you will have successfully integrated advanced beauty effects.

iOS

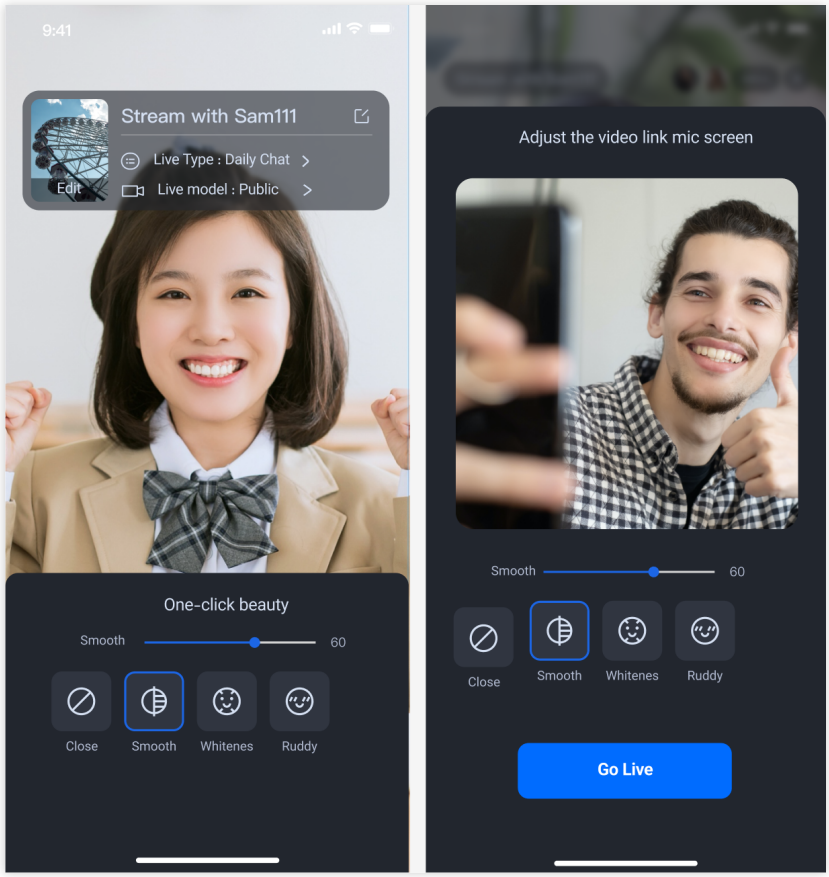
Last updated：2024-08-02 16:39:02

TUILiveKit offers two types of beauty effects: `Basic Beauty` and `Advanced Beauty`. If you are not satisfied with the results of Basic Beauty, you can choose to integrate Advanced Beauty to meet your more advanced beauty needs.

Basic Beauty

TUILiveKit comes with Basic Beauty functionality by default. Basic Beauty includes features such as skin whitening, skin smoothing, and adding a ruddy tint to the complexion. You can adjust the intensity of these beauty effects to meet different requirements. These features are already built-in within TUILiveKit, **so there is no need for additional configuration or integration.**

Panel Display



Advanced Beauty

TUILiveKit utilizes Tencent Effects Beauty for advanced beauty effects.

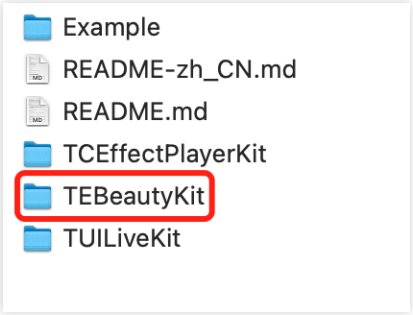
Effect Showcase

Chin slimming	Eye distance	Nose slimming
		

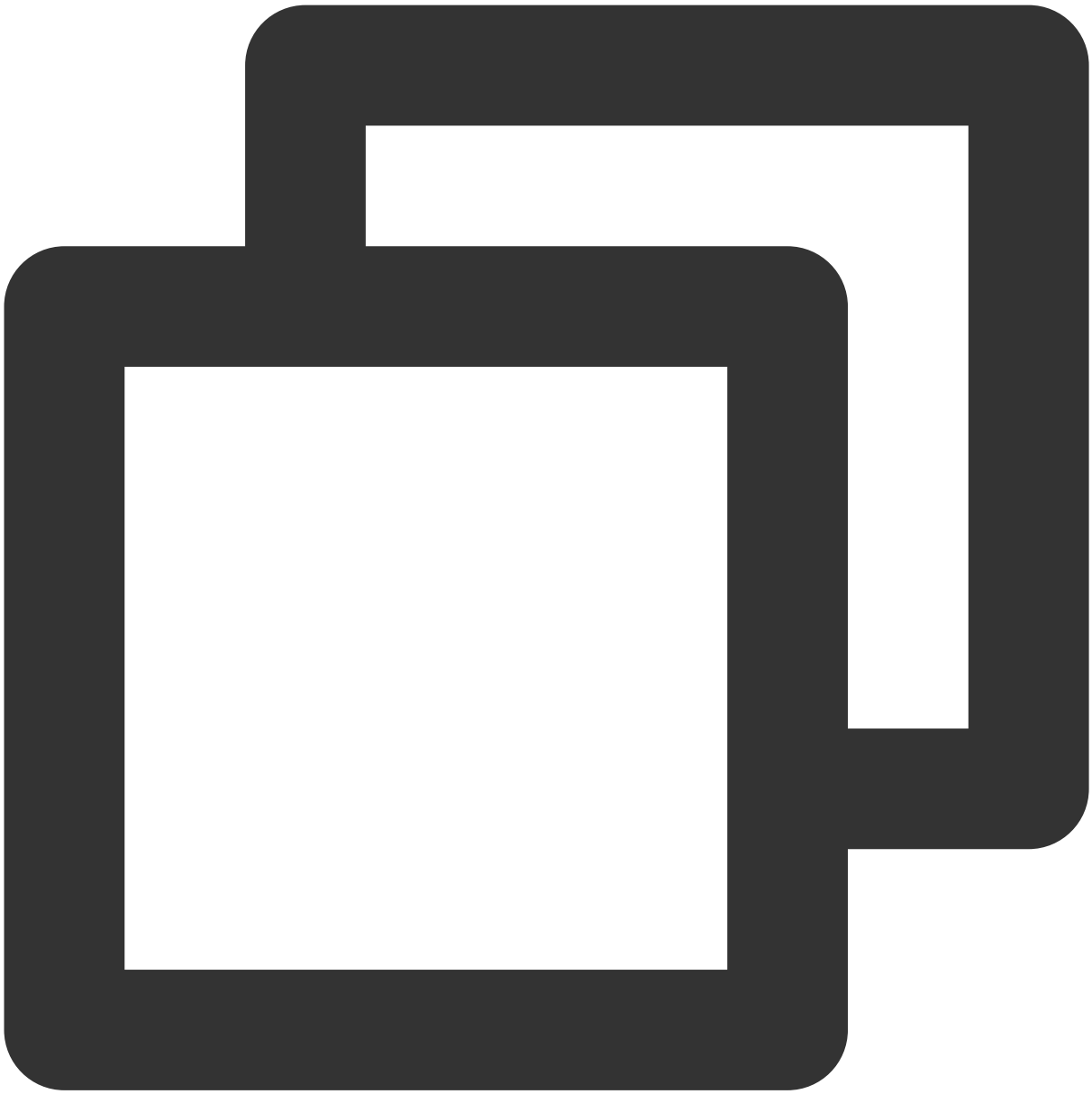
Integration Guide

Step 1: Integrating `TEBeautyKit`

1. Download and extract TUILiveKit. Copy the `iOS/TEBeautyKit` folder to your project, at the same level as the `Podfile` folder.



2. Please edit the Podfile and add the following code:



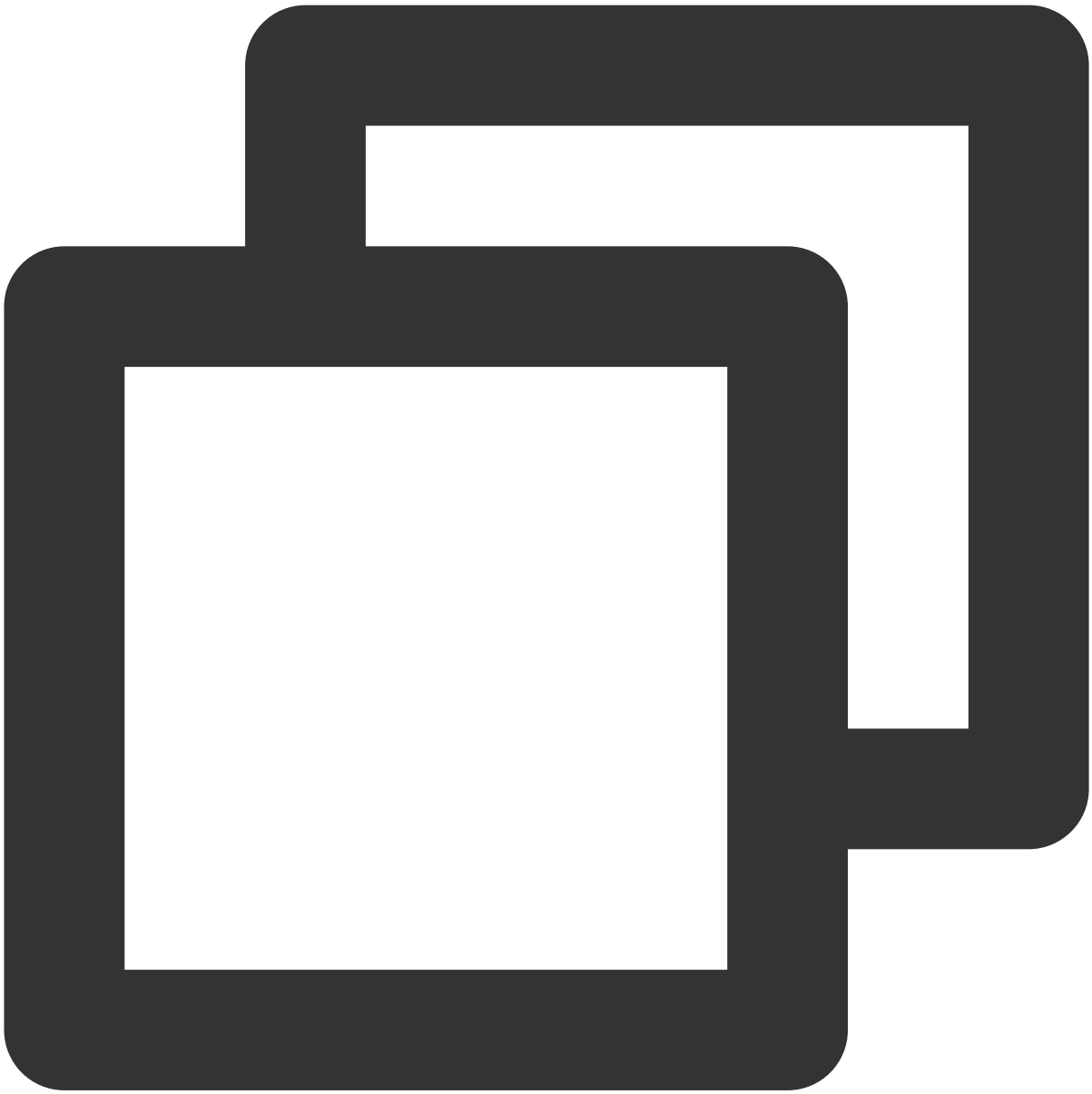
```
pod 'TEBeautyKit',:podspec => './TEBeautyKit/TEBeautyKit.podspec'
```

3. Save the changes and run `pod install` in the terminal to install the tebeautykit dependency.

Step 2: Authorization & Setting Beauty Resources

1. Apply for authorization and obtain `LicenseURL` and `LicenseKEY` . Please refer to the License Guide for more information.

2. Set the `URL` and `KEY` in the initialization code of the relevant business module, and configure the beauty filter resources. For example, on iOS, you can set the relevant content in the `didFinishLaunchingWithOptions` method of `AppDelegate` .



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

import TEBeautyKit
```

```
func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions: [UIApplication.Launch
        TEBeautyKit.setTELicense("LicenseURL", key: "LicenseKEY") { code, message i
            TEUIConfig.shareInstance().setPanelLevel(.S1_07) // Replace S1_07 with
        }
        return true
}
```

Note:

If you are unsure about the beauty package number, click here to view the overview of beauty package numbers.

By completing the aforementioned steps, you will have successfully integrated advanced beauty effects.

Client APIs (TUICallKit)
iOS
UIKit API

Last updated：2024-07-05 19:42:58

Introduction

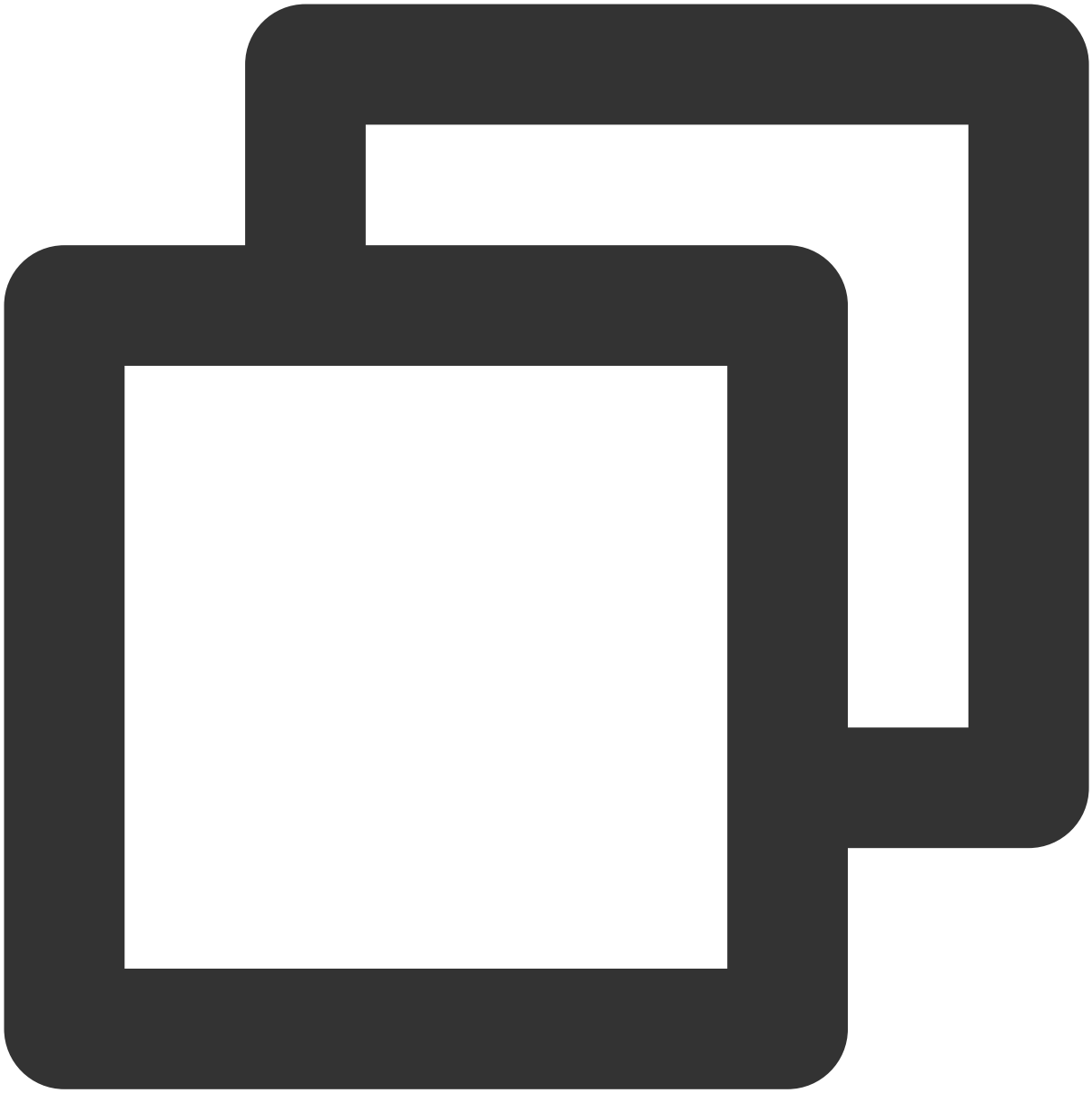
TUILiveKit is an open source UI suite for multi-person video broadcast layer. At present, the iOS platform supports Swift language, and the live broadcast UI can be aroused through simple API calls.

TUILiveRoomAnchorViewController

API	describe
init	constructing an anchor object for live broadcasting

init

Initialize `TUILiveRoomAnchorViewController` object.



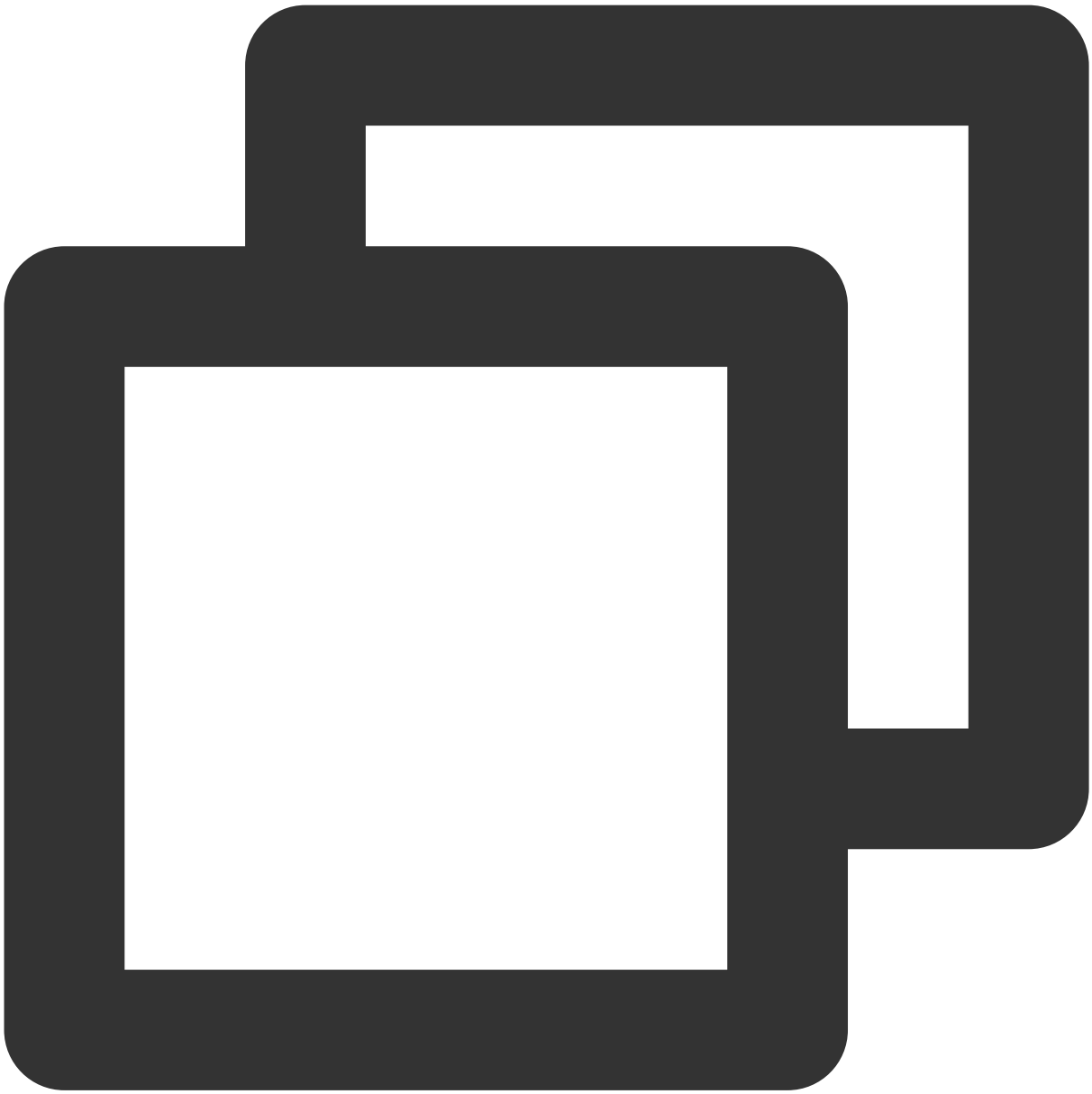
```
public init(roomId:String)
```

TUILiveRoomAudienceViewController

API	describe
init	building a viewer pull-stream object

init

Initialize `TUILiveRoomAudienceViewController` object.



```
public init(roomId:String)
```

Engine API
API Overview

Last updated：2024-07-05 19:46:05

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 19:47:17

Introduction

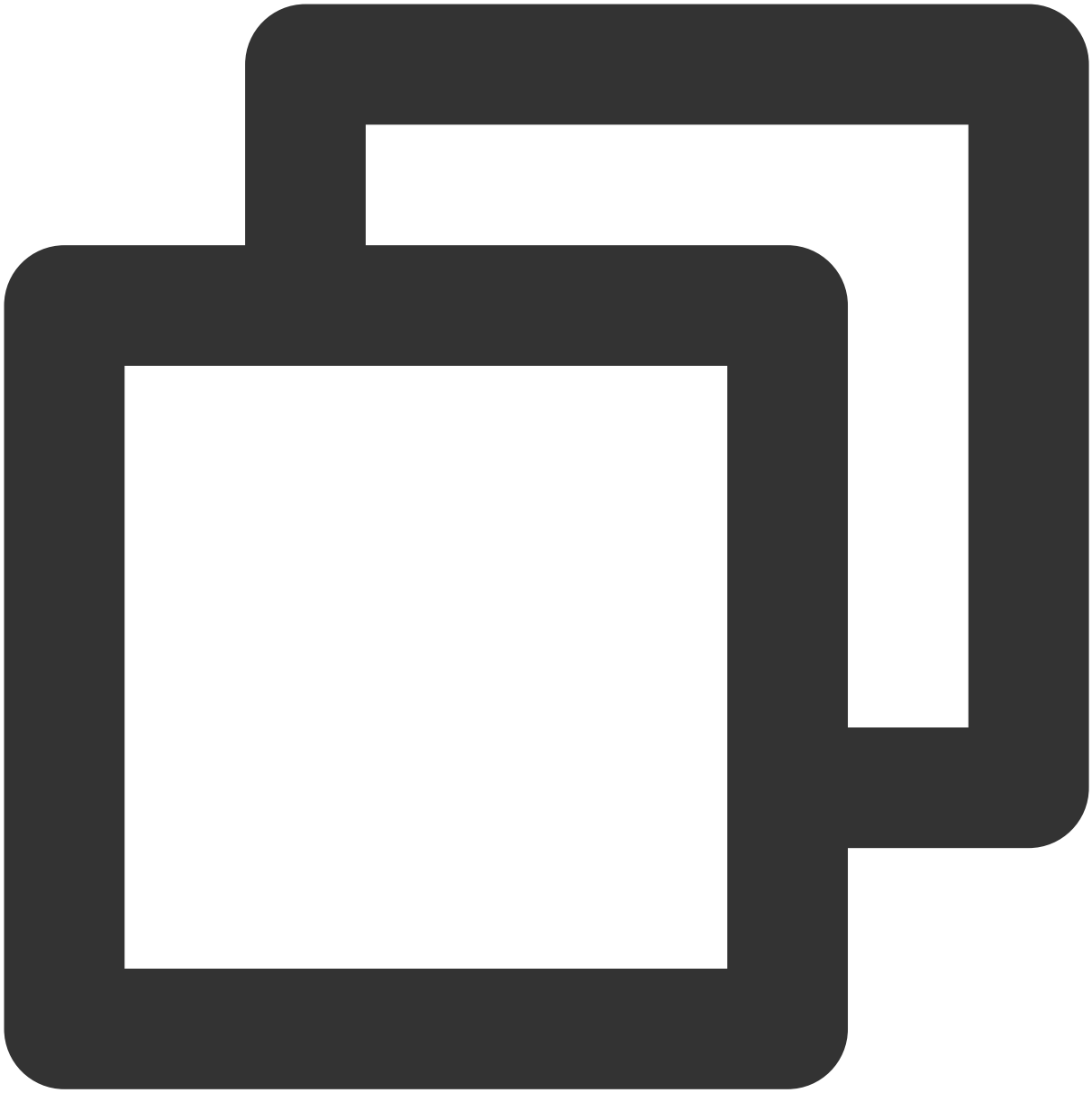
TUILiveKit is an open source UI suite for multi-person video broadcast layer. At present, the java platform supports Swift language, and the live broadcast UI can be aroused through simple API calls.

TUILiveRoomAnchorFragment Class

API	describe
TUILiveRoomAnchorFragment(String roomId)	constructing an anchor object for live broadcasting

TUILiveRoomAnchorFragment

constructing an anchor object for live broadcasting



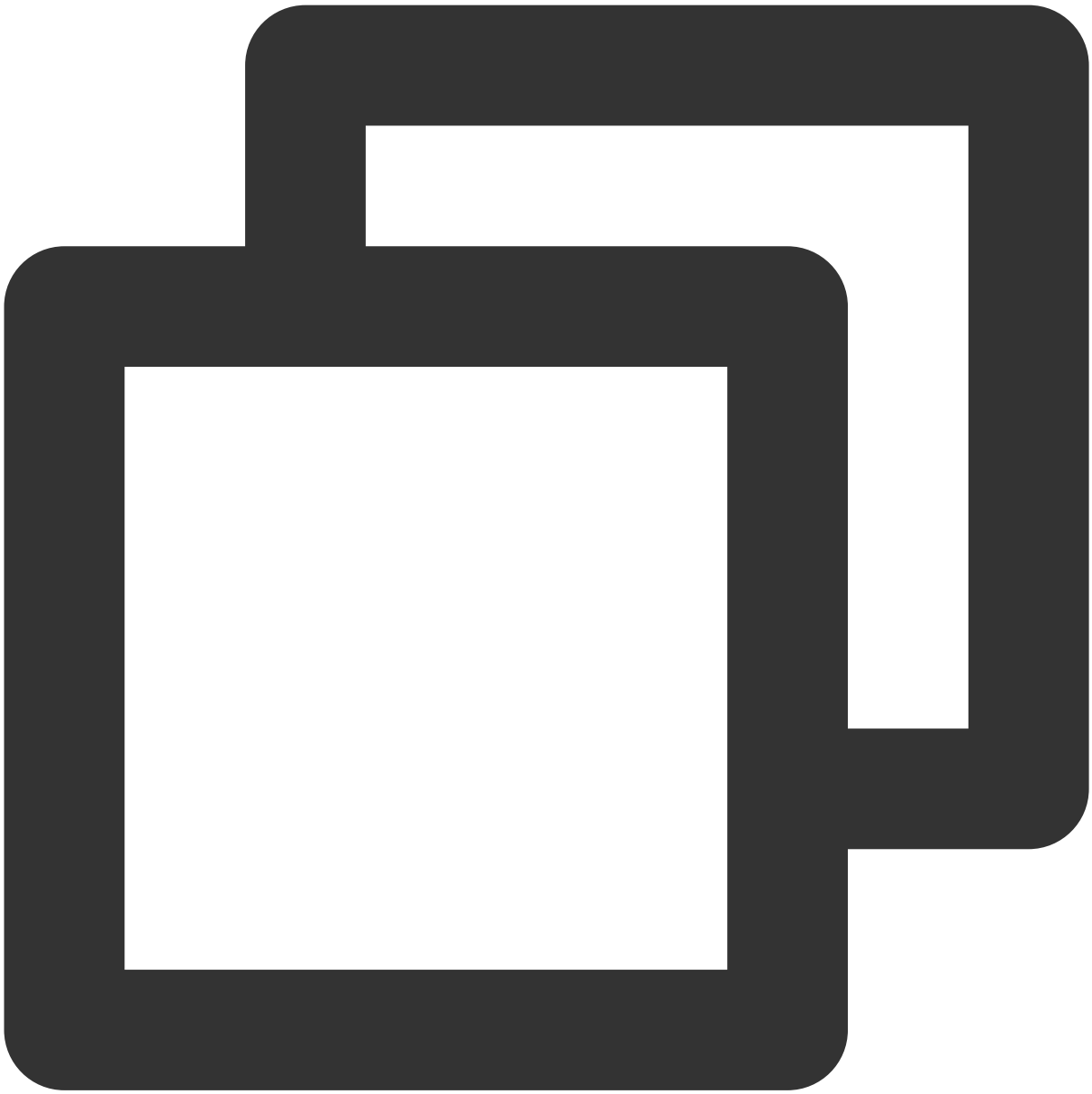
```
public TUILiveRoomAnchorFragment(roomId:String)
```

TUILiveRoomAudienceFragment Class

API	describe
TUILiveRoomAudienceFragment(String roomId)	building a viewer pull-stream object

TUILiveRoomAudienceFragment

building a viewer pull-stream object



```
public TUILiveRoomAudienceFragment(roomId:String)
```

Engine API
API Overview

Last updated：2024-07-05 19:51:31

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：2024-05-09 19:18:36

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：2024-04-26 14:30:52

April 2024

Version 1.0.0 Released In April 22, 2024

Post an update	describe	Release time
Version 1.0.0	Supports live video streaming Supports custom gifts Supports custom bullet comments Supports beauty filters Supports background music	2024.04.22

Android

Last updated：2024-04-26 14:30:52

April 2024

Version 1.0.0 Released In April 22, 2024

Post an update	describe	Release time
Version 1.0.0	Supports live video streaming Supports custom gifts Supports custom bullet comments Supports beauty filters Supports background music	2024.04.22

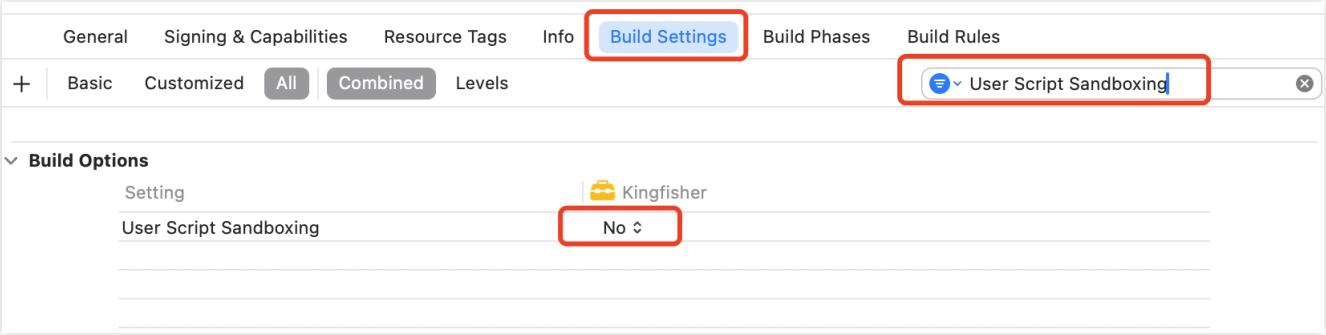
FAQs（TUILiveKit）
iOS

Last updated：2024-05-17 11:48:33

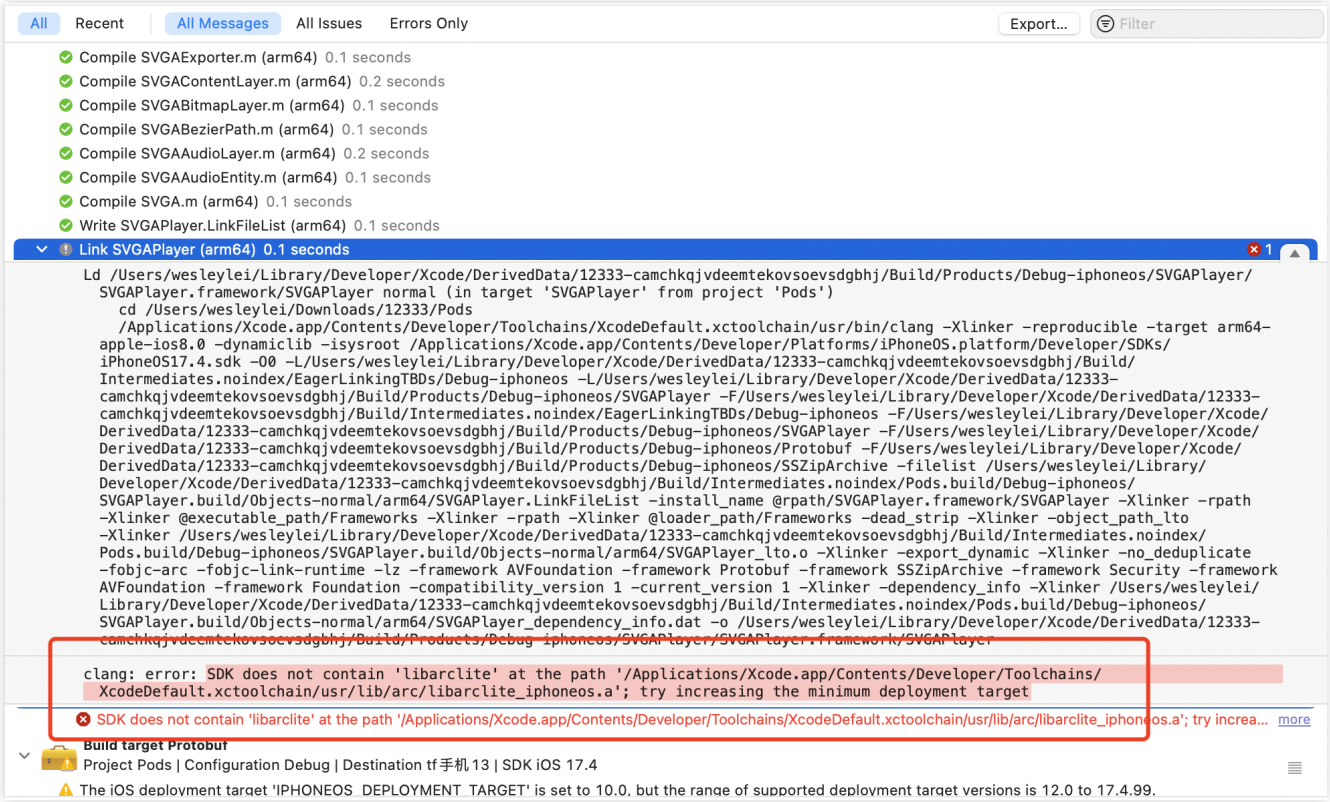
Xcode 15 compiler error?

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

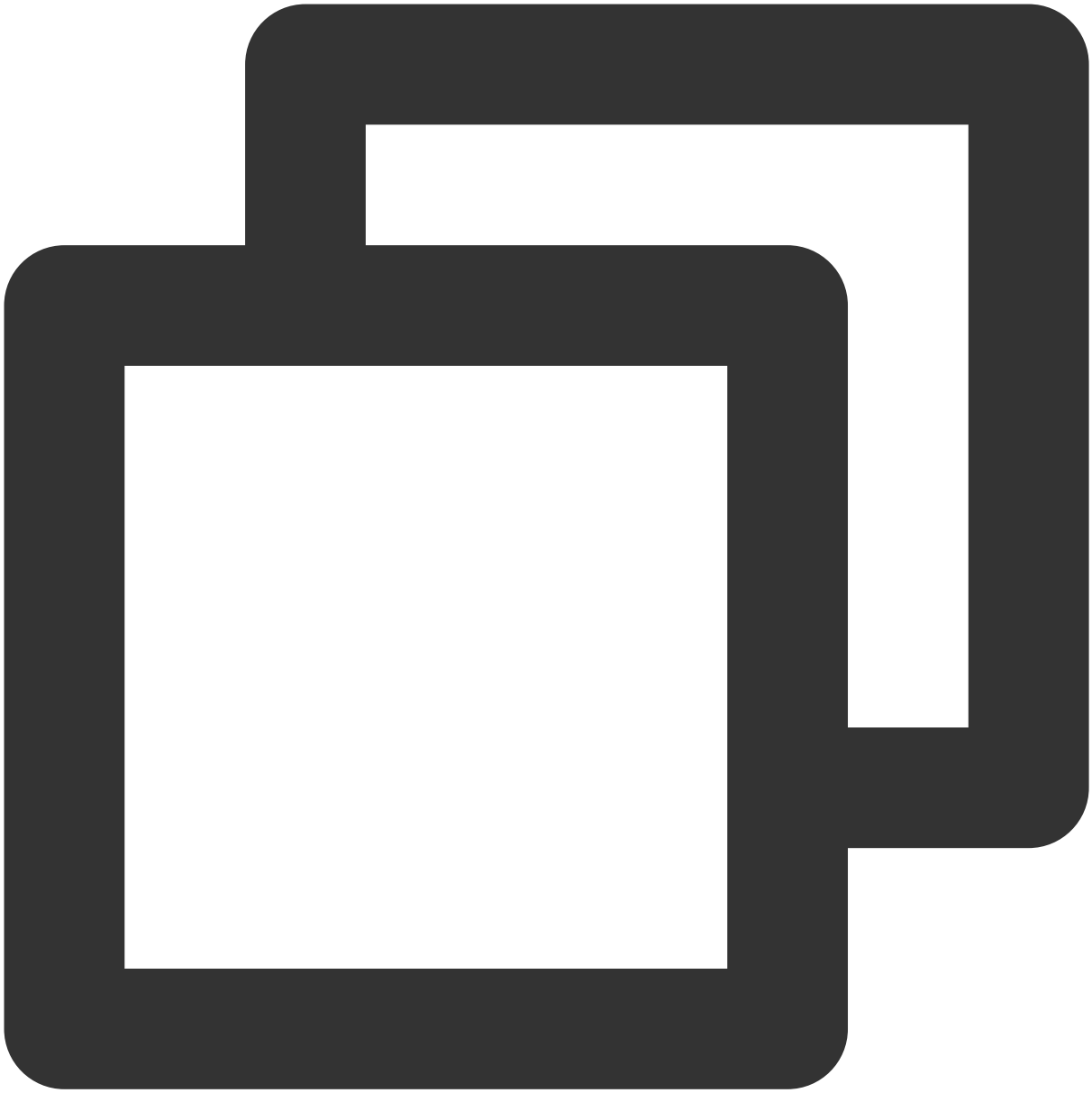
You can set User Script Sandboxing to **NO** in **Build Settings:**



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

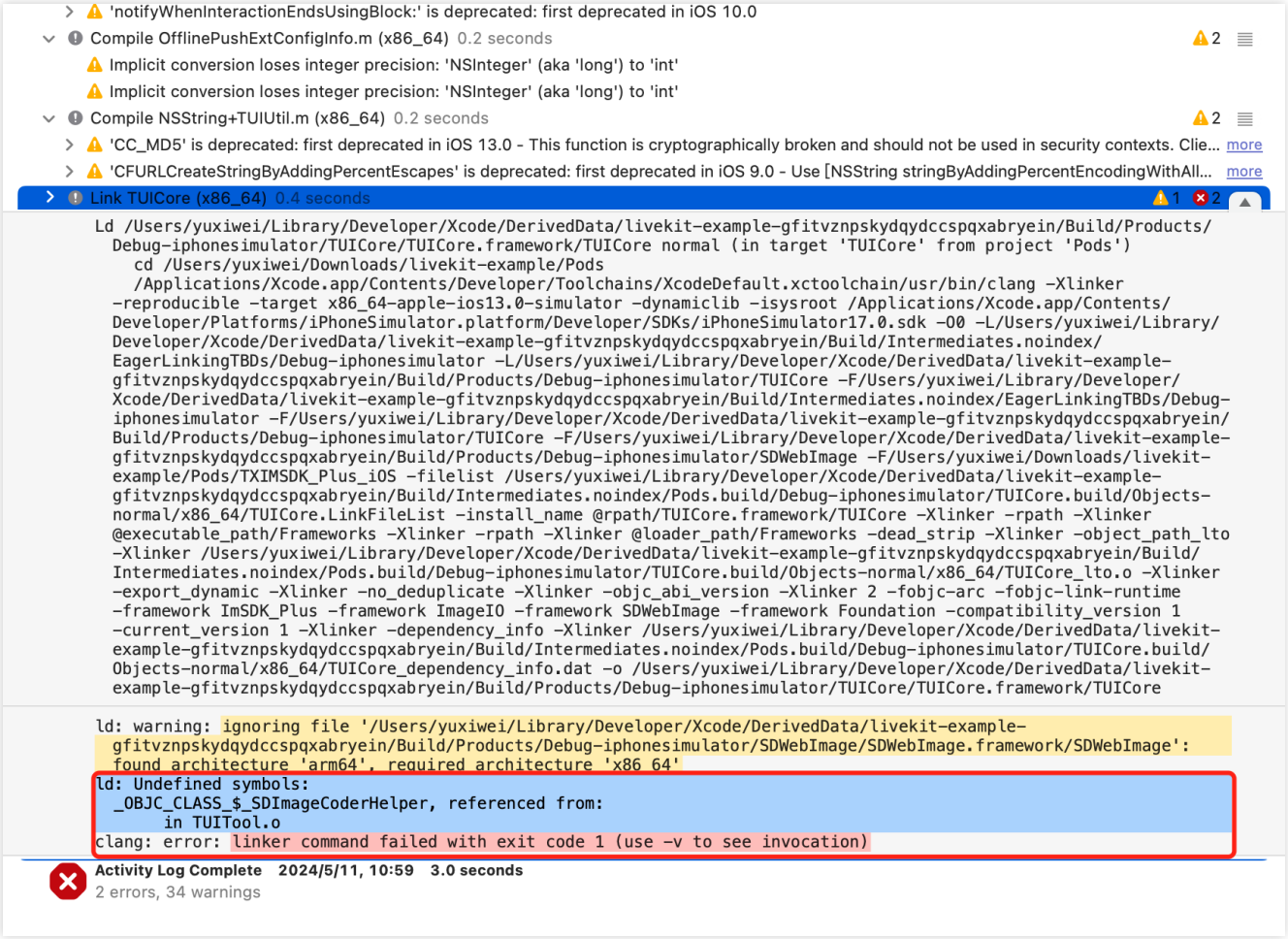


Add the following code to the Podfile:

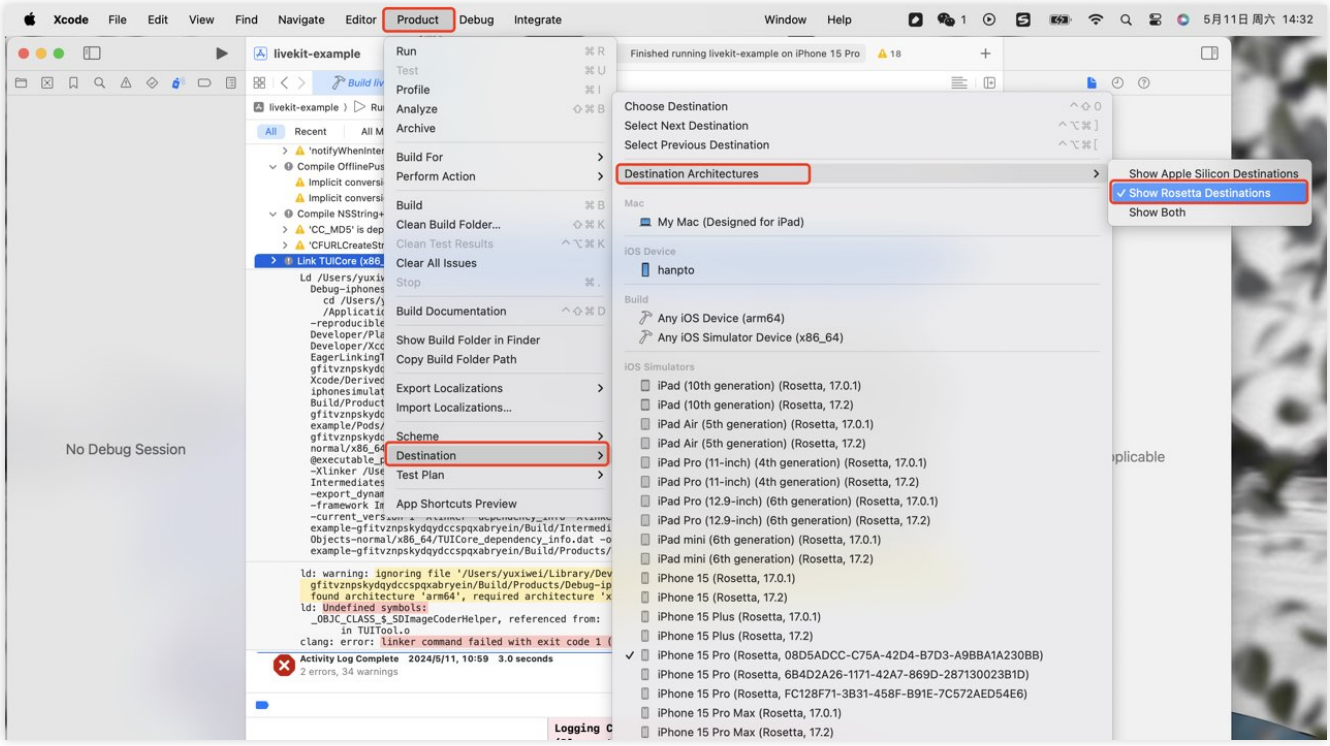


```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '13.0'
    end
  end
end
```

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.



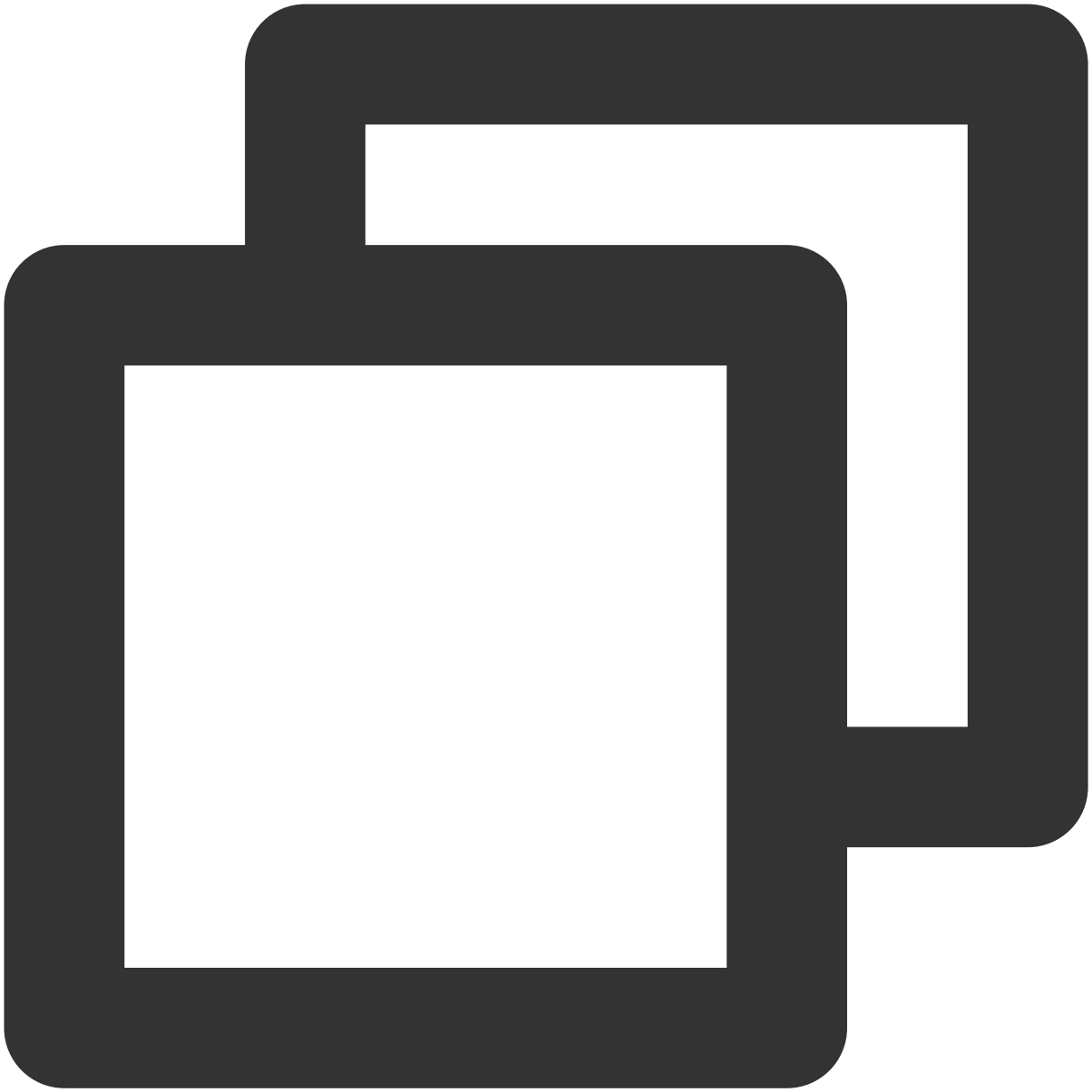
The xcode configuration needs to be modified. xcode open projects > **Product > Destination > Destination Architectures** can choose which mode of emulator to open with, and need to select the ending emulator (**Rosetta**).



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

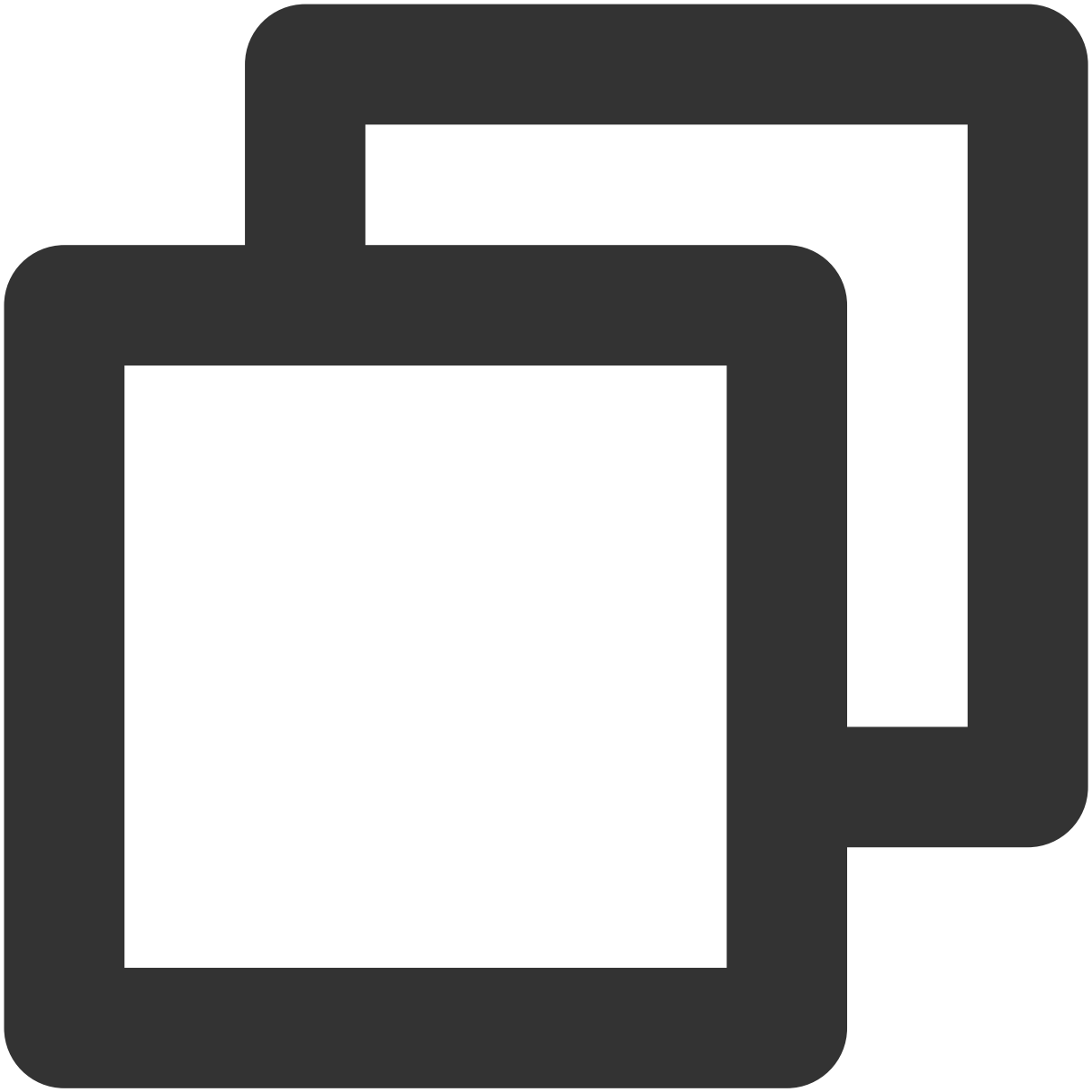
Tencent Cloud's audio and video libraries cannot be integrated at the same time, and there may be symbol conflicts. You can handle it according to the following scenarios.

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'
```

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：2024-07-03 11:22:38

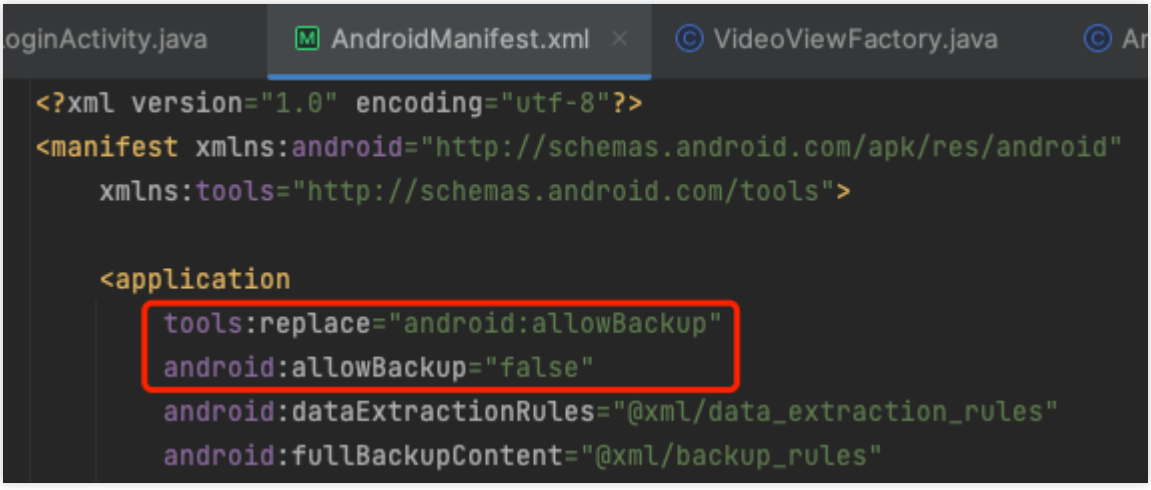
Can TUILiveKit use TRTC without introducing IM SDK?

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.

allowBackup exception, How to Handle?

```
Manifest merger failed : Attribute application@allowBackup value=(false) from AndroidManifest.xml:7:9-36
    is also present at [com.github.yyued:SVGAPlayer-Android:2.6.1] AndroidManifest.xml:12:9-35 value=(true).
    Suggestion: add 'tools:replace="android:allowBackup"' to <application> element at AndroidManifest.xml:5:5
```

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 need to use a Theme.AppCompat theme?

```
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.uikit.
 .LoginActivity}: java.lang.IllegalStateException: You need to use a Theme.AppCompat theme (or descendant) with th
    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>
```

Failed to open the web page address in the browser?

Solution: You can add the following configurations to the AndroidManifest.xml file of your project:

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <queries>
        <intent>
            <action android:name="android.intent.action.VIEW" />
            <data android:scheme="https" />
        </intent>
    </queries>
```