Tencent Cloud

Chat

Get Started

Product Documentation

Contents

Get Started
Android

Last updated：2023-09-19 15:20:02

Directions

Step 1. Create an app

1. Log in to the Chat console.

Note

If you already have an app, record its SDKAppID and obtain key information.

A Tencent Cloud account can create a maximum of 300 Chat apps. If you want to create a new app, disable and delete an unwanted app first. **Once an app (along with its SDKAppID) is deleted, the service it provides and all its data are lost. Proceed with caution**.

2. Click **Create Application**, enter your app name, and click **Confirm**.

3. After creation, you can see the status, service version, SDKAppID, creation time, tag, and expiry time of the new app on the overview page of the console. Record the SDKAppID.

Step 2. Obtain key information

1. Click the target app card to go to the basic configuration page of the app.

2. In the **Basic Information** area, click **Display key**, and then copy and save the key information.

Caution

 Store the key information properly to prevent disclosure.

Step 3. Download and configure the demo source code

1. Download the Chat demo project. For more download information, see SDK Download.

Note

To respect the copyright of emoji design, the downloaded demo project does not contain sliced images of major emoji elements. You can use your local emoji packs to configure code. Unauthorized use of the emoji pack in the Chat demo may infringe on the design copyright.

2. Open the project in the terminal directory and find the file `GenerateTestUserSig` in `Android/Demo/app/src/main/java/com/tencent/qcloud/tim/demo/signature/GenerateTestUserSig.java`.

3. Set relevant parameters in the `GenerateTestUserSig` file:

SDKAPPID: set it to the SDKAppID obtained in Step 1.

SECRETKEY: enter the key obtained in Step 2.

```
/**
 * Tencent Cloud SDKAppID. Set it to the SDKAppID of your account.
 * <p>
 * You can view your SDKAppID after creating an application in the
 * [Tencent Cloud IM console](https://console.intl.cloud.tencent.com/im).
 * SDKAppID uniquely identifies a Tencent Cloud account.
 */
public static final int SDKAPPID = 0;


/**
 * Signature validity period, which should not be set too short
 * <p>
 * Time unit: second
 * Default value: 604800 (7 days)
 */
private static final int EXPIRETIME = 604800;


/**
 * Follow the steps below to obtain the key required for UserSig calculation.
 * <p>
 * Step 1. Log in to the [Tencent Cloud IM console](https://console.intl.cloud.tencent.com/im),
 * and create an application if you don't have one.
 * Step 2. Click Application Configuration to go to the basic configuration page and locate
 * Account System Integration.
 * Step 3. Click View Key to view the encrypted key used for UserSig calculation. Then copy and
 * paste the key to the variable below.
 * <p>
 * Note: this method is for testing only. Before commercial launch, please migrate the UserSig
 * calculation code and key to your backend server to prevent key disclosure and traffic stealing.
 * Reference: https://intl.cloud.tencent.com/document/product/1047/34385
 */
private static final String SECRETKEY = "";
```

App Information

SDKAppID	300██
Application Name	
Application Type	Game
Application Introduction	-

Basic Information

Key	****** Display ke Key information
Creation Time	2022-01-13
Last Modified	2022-01-13

Caution

In this document, the method to obtain `UserSig` is to configure a `SECRETKEY` in the client code. In this method, the `SECRETKEY` is vulnerable to decompilation and reverse engineering. Once your `SECRETKEY` is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and

provide an app-oriented API. When `UserSig` is needed, your app can send a request to the business server to obtain a dynamic `UserSig`. For more information, see How to Generate UserSig on the Server.

Step 4. Compile and run the demo

Import the demo project with Android Studio, and then compile and run it.

For more information, see the file `README.md` in the corresponding directory of the demo project cloned in Step 3.

Environment requirements

Android Studio-Chipmunk

Gradle-6.7.1

Android Gradle Plugin Version-4.2.0

kotlin-gradle-plugin-1.5.31

Caution

The demo is integrated with the audio/video call feature by default. However, the TRTC SDK on which the audio/video call feature relies currently does not support simulators. Please use real devices for demo running or debugging.

iOS

Last updated：2023-09-19 15:22:07

This document describes how to quickly run the IM demo for iOS.

Directions

Step 1. Create an app
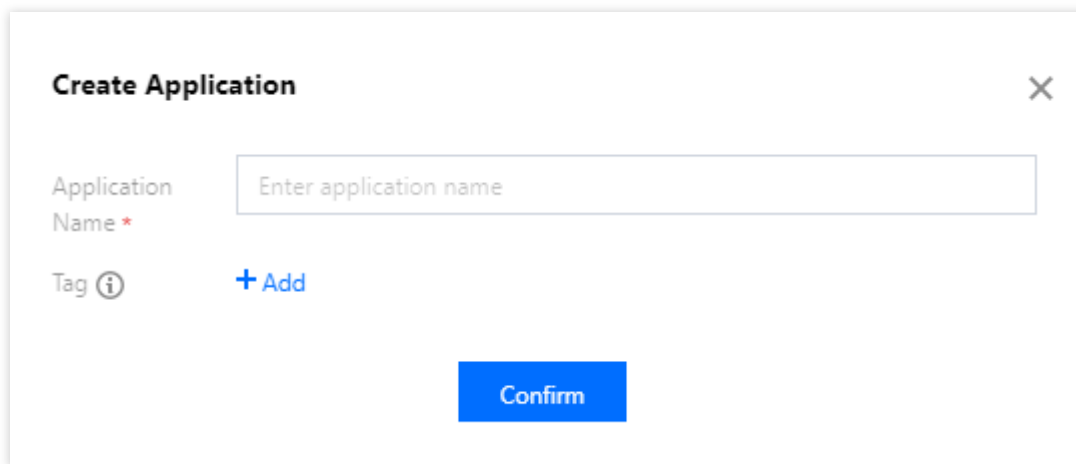
1.1 Log in to the IM console.

Note

If you already have an app, record its SDKAppID and obtain key information.

A Tencent Cloud account can create a maximum of 300 IM apps. If you want to create a new app, disable and delete an unwanted app first. **Once an app (along with its SDKAppID) is deleted, the provided service and all its data are lost. Please proceed with caution**.

1.2 Click **Create Application**, enter your app name, and click **Confirm**.

1.3 After creation, you can see the status, service version, SDKAppID, creation time, tag, and expiry time of the new app on the overview page of the console. Record the SDKAppID.

Tencent Cloud

Step 2. Obtain key information

1.1 Click the target app card to go to the basic configuration page of the app.

1.2 In the **Basic Information** area, click **Display key**, and then copy and save the key information.

Caution

 Please store the key information properly to prevent disclosure.

Step 3. Download and configure the demo source code

1.1 Download the IM demo project. For more information, see SDK and Demo Source Code.

Note

To respect the copyright of emoji design, the downloaded demo project does not contain sliced images of major emoji elements. You can use your local emoji packs to configure code. Unauthorized use of the emoji pack in the IM demo may infringe on the design copyright.

1.2 Open the project in the terminal directory and find the `GenerateTestUserSig` file in the following paths:

iOS: iOS/Demo/TUIKitDemo/Private/GenerateTestUserSig.h

macOS: Mac/Demo/TUIKitDemo/Debug/GenerateTestUserSig.h

1.3 Set relevant parameters in the `GenerateTestUserSig` file:

SDKAPPID: Set it to the SDKAppID obtained in Step 1.

SECRETKEY: Enter the key obtained in Step 2.

Caution

In this document, the method to obtain `UserSig` is to configure a `SECRETKEY` in the client code. In this method, the `SECRETKEY` is vulnerable to decompilation and reverse engineering. Once your `SECRETKEY` is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an app-oriented API. When `UserSig` is needed, your app can send a request to the business server to obtain a dynamic `UserSig`. For more information, see Generating UserSig on the Server.

Tencent Cloud

Step 4. Compile and run the demo

See the file `README.md` in the corresponding directory of the demo project cloned in Step 3.

1.1 Run the following command on the terminal to check the pod version:
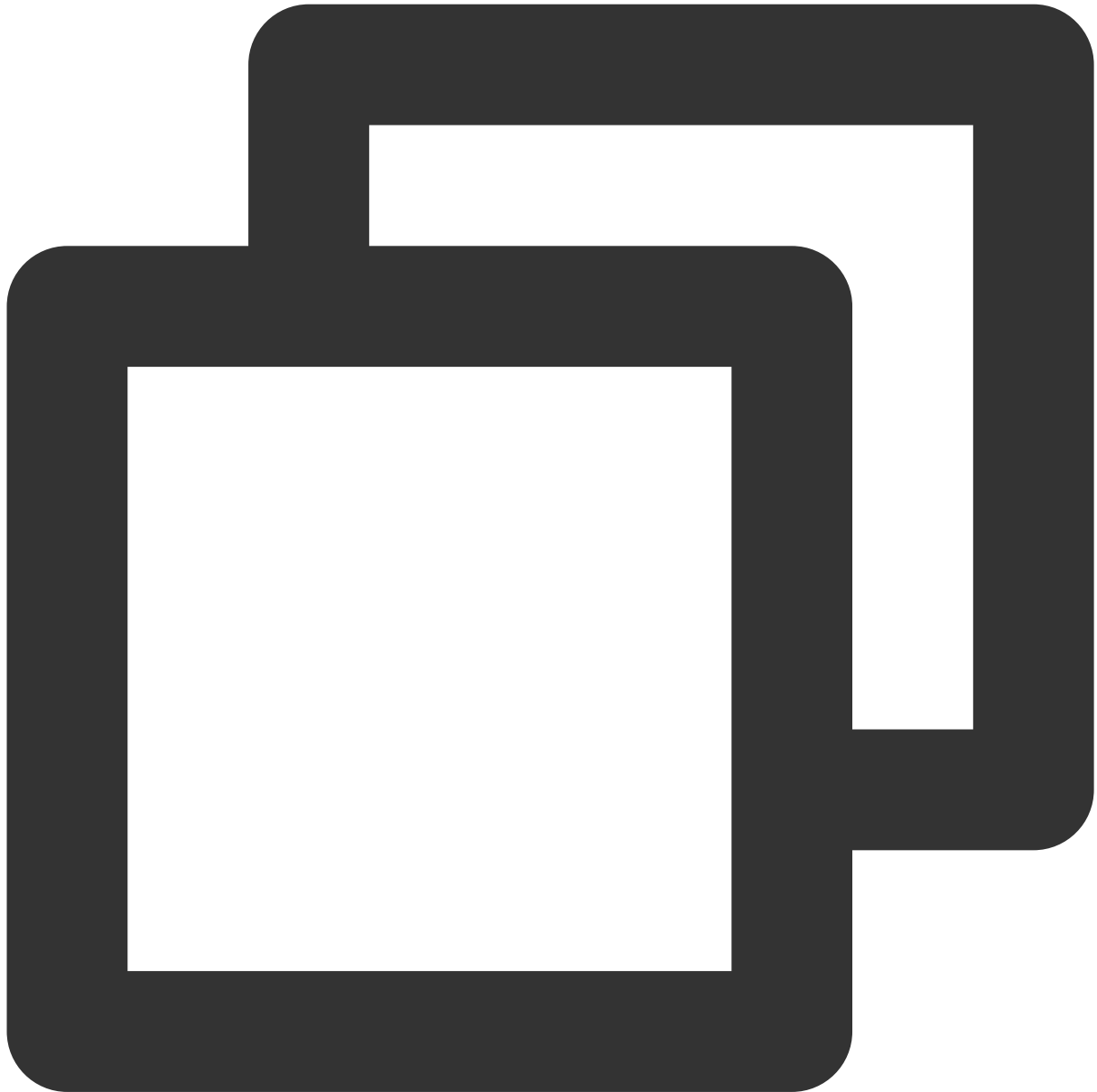

```
pod --version
```

If the system indicates that no pod exists or that the pod version is earlier than 1.7.5, run the following commands to install the latest pod.
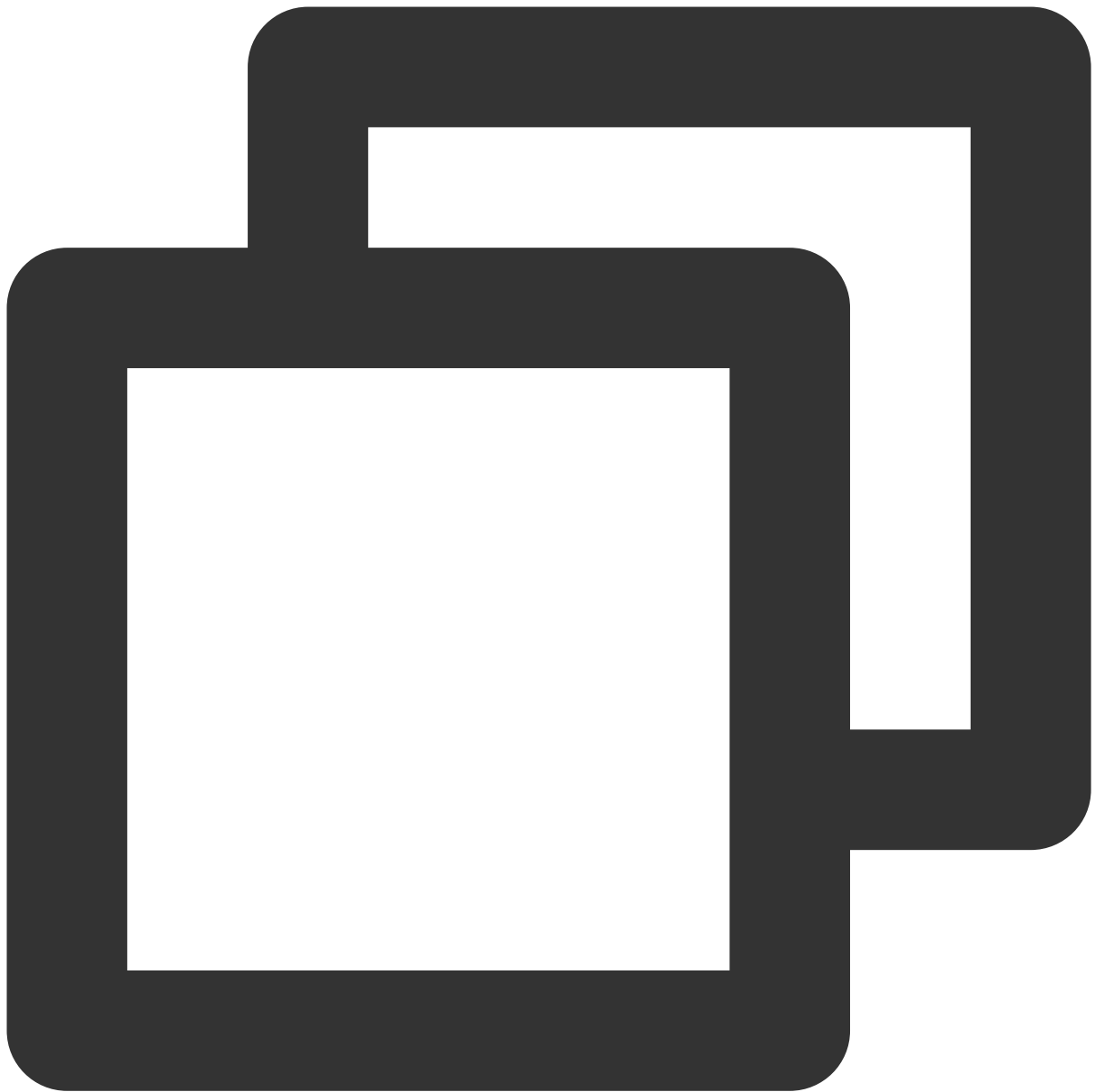
```
// Change gem sources.
gem sources --remove https://rubygems.org/
gem sources --add https://gems.ruby-china.com/
// Install pod.
sudo gem install cocoapods -n /usr/local/bin
// If multiple versions of Xcode are installed, run the following command to choose
sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer
// Update the local pod library.
pod setup
```

1.2 Run the following commands on the terminal to install dependent libraries:
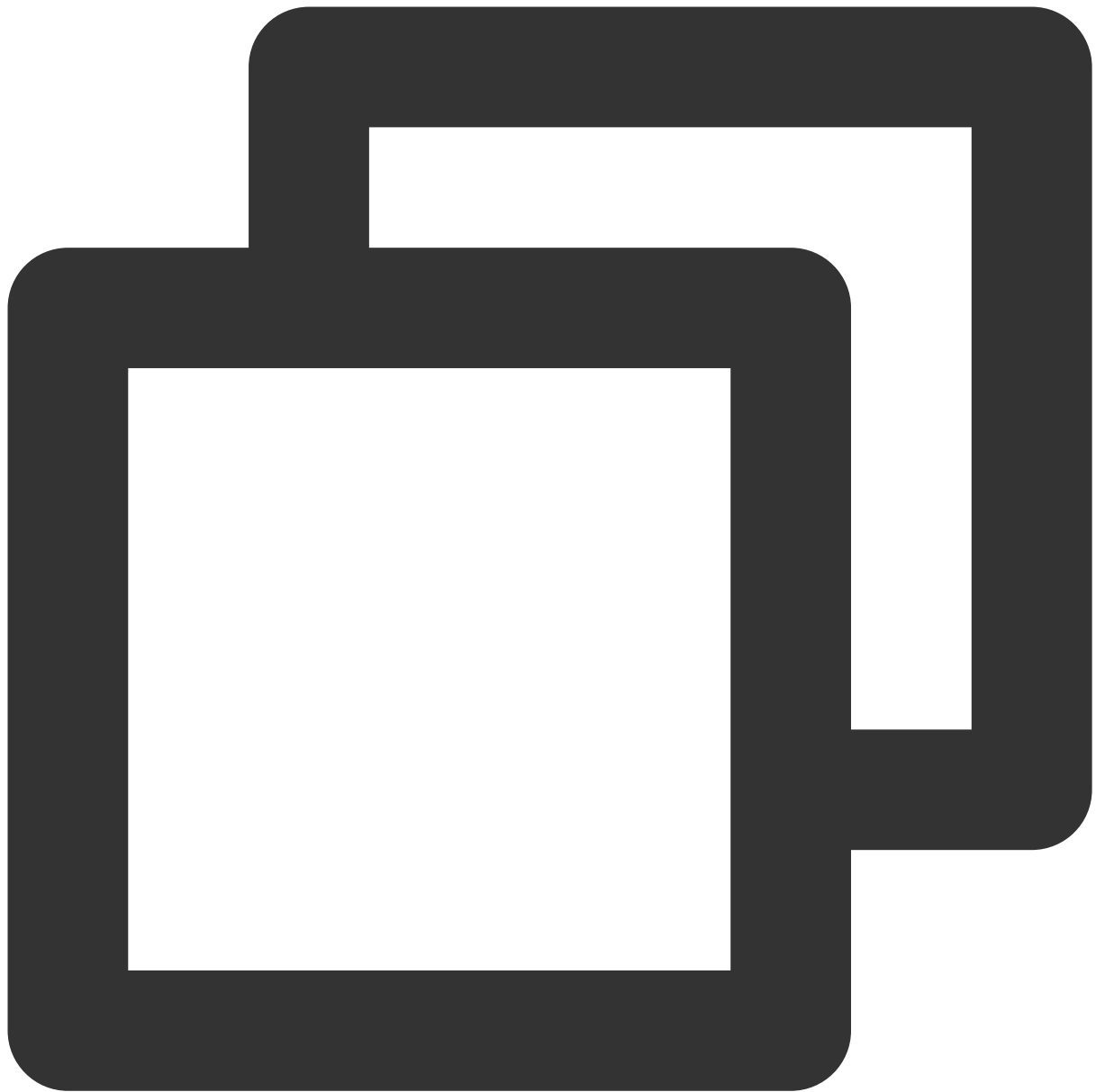
```
// iOS
cd iOS/TUIKitDemo
pod install
// macOS
cd Mac/TUIKitDemo
pod install
```

If installation fails, run the following command to update the local CocoaPods repository list:

```
pod repo update
```

1.3 Compile and run the demo:

iOS: Go to the iOS/TUIKitDemo folder, and open `TUIKitDemo.xcworkspace` to compile and run the demo.

macOS: Go to the Mac/TUIKitDemo folder, and open `TUIKitDemo.xcworkspace` to compile and run the demo.

Caution

The demo is integrated with the audio/video call feature by default. However, the TRTC SDK on which the audio/video call feature relies currently does not support simulators. Please use real devices for demo running or debugging.

Advanced Features

[TUIKit](#)

[Enabling Video Calls](#)

References

[Pricing](#)

Web React

Last updated：2024-04-01 16:36:36

Chat UIKit is a UI component library based on Tencent Cloud IM SDK. It provides universal UI components to offer features such as conversation, chat, relationship chain, group, and audio/video call features. With these UI components, you can quickly build your own business logic.

When implementing UI features, components in Chat UIKit will also call the corresponding APIs of the IM SDK to implement IM-related logic and data processing, allowing developers to focus on their own business needs or custom extensions.

Click to experience the Demo.
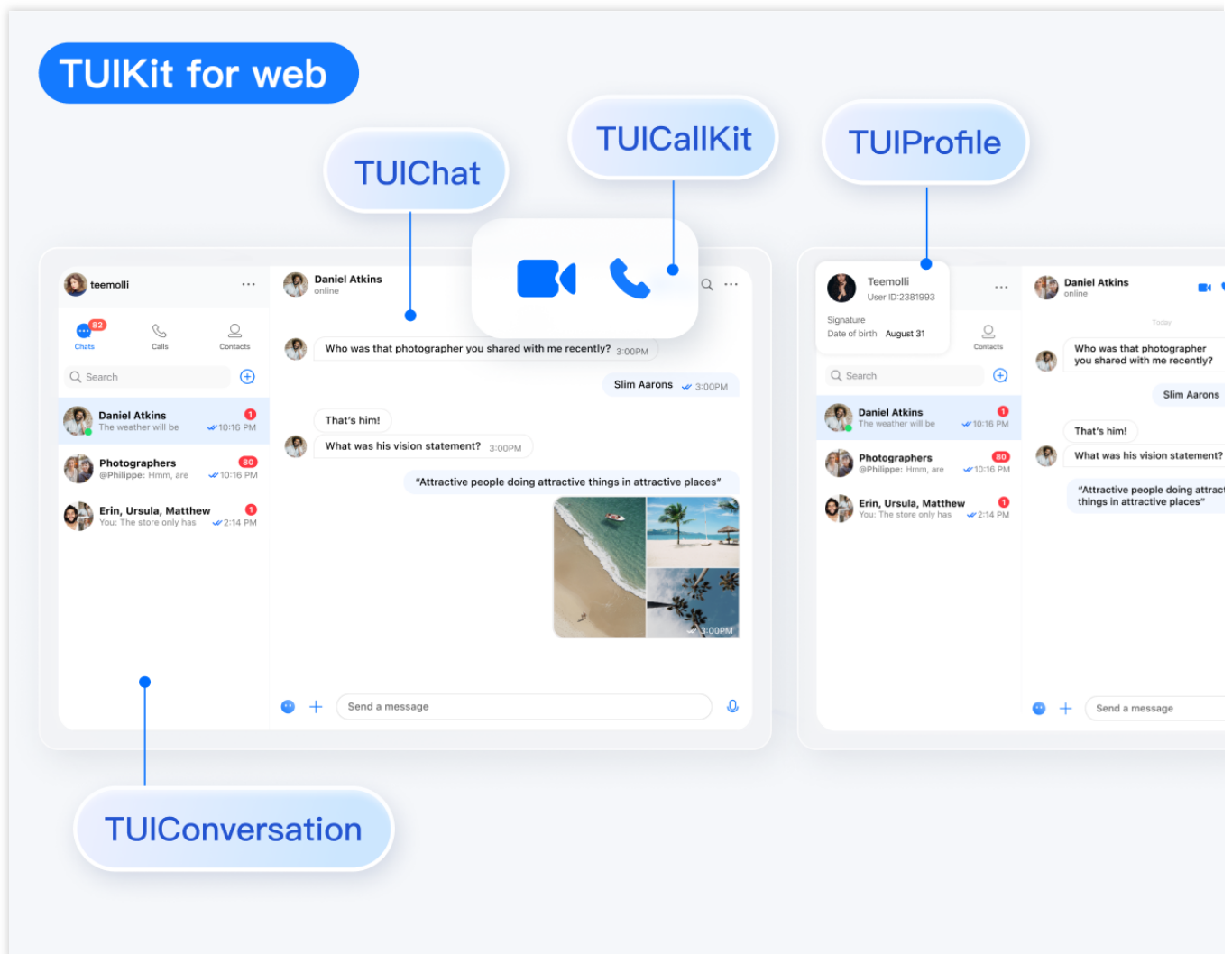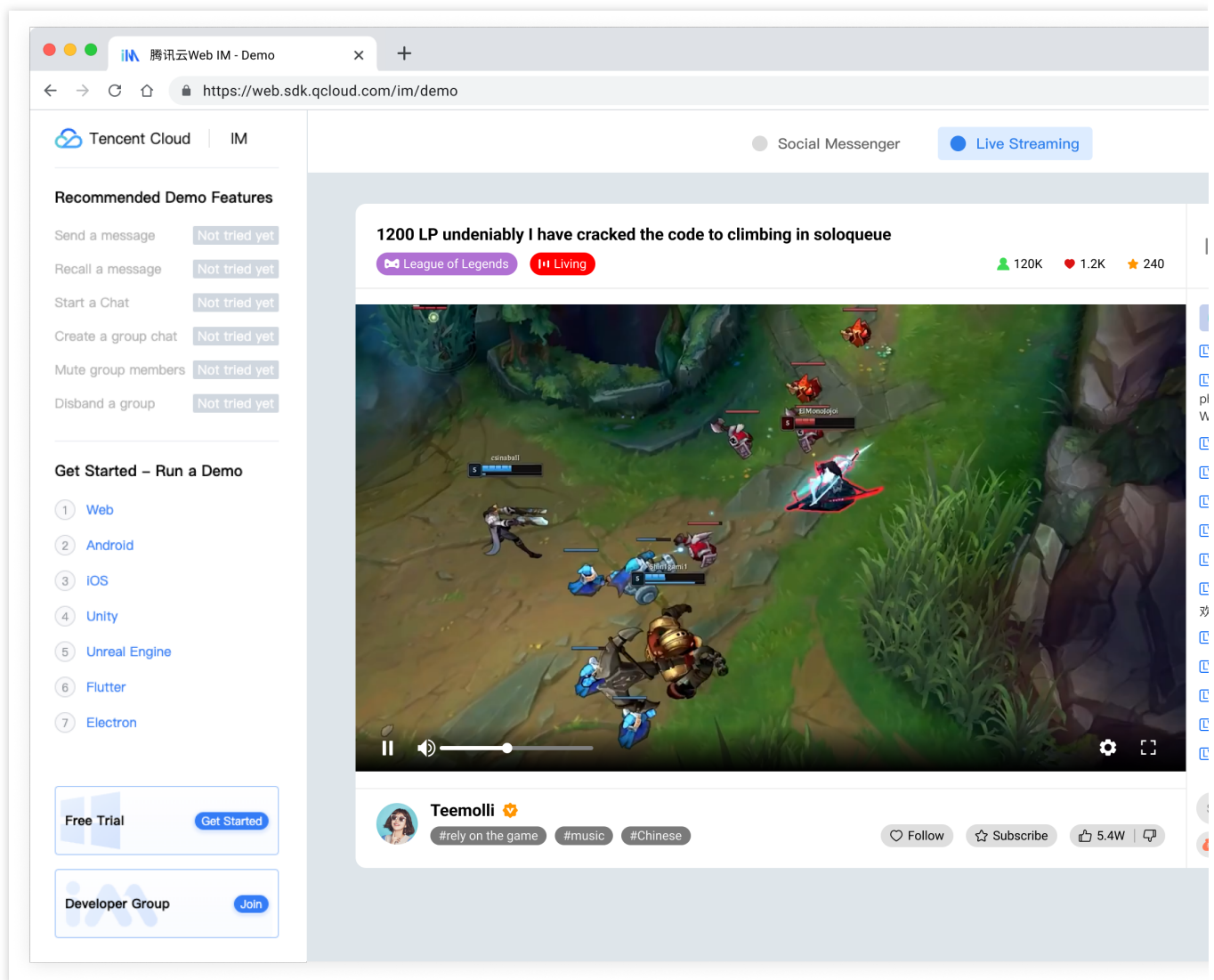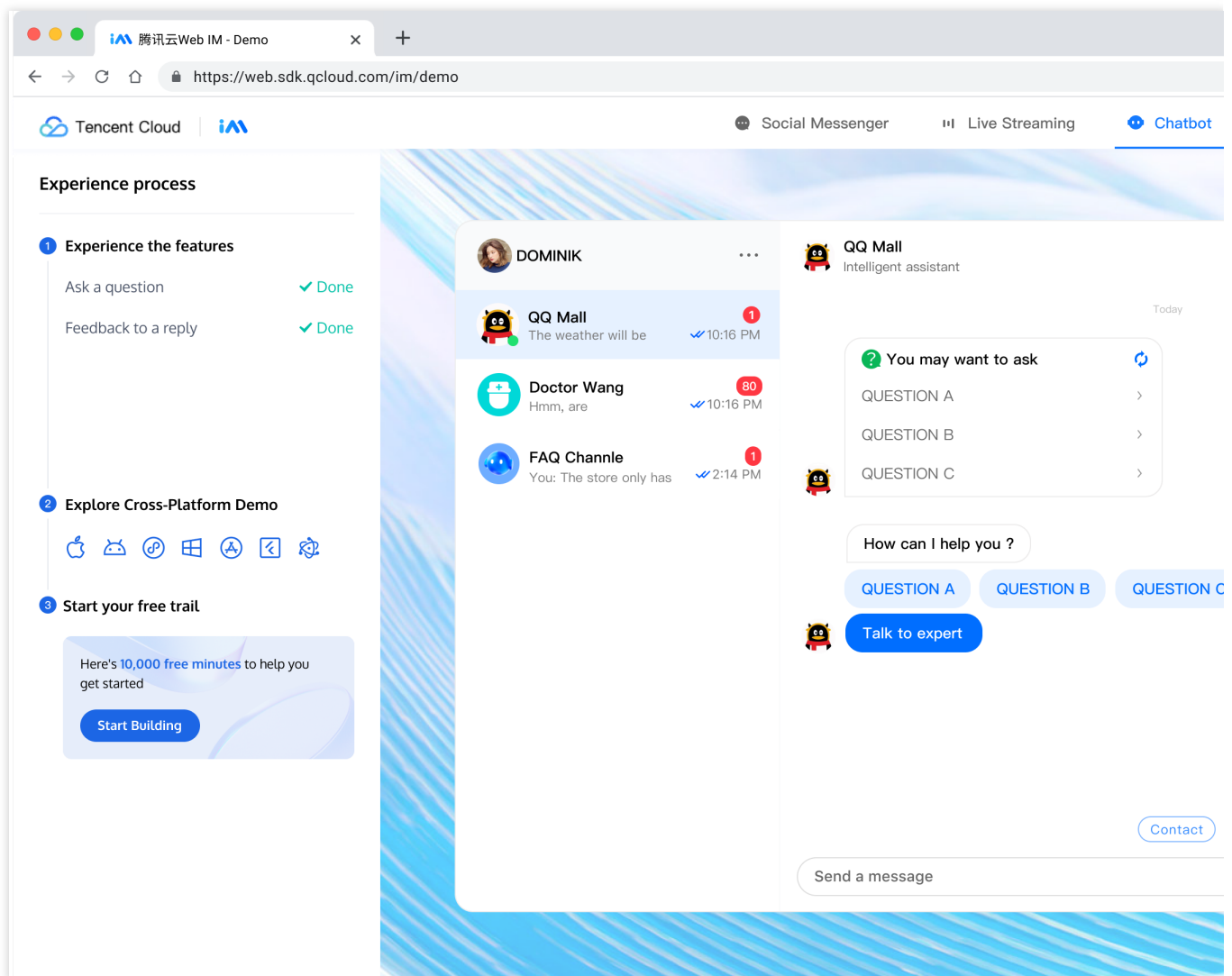
Tencent Cloud

Running Demo

Step 1: Download the source code

MacOS
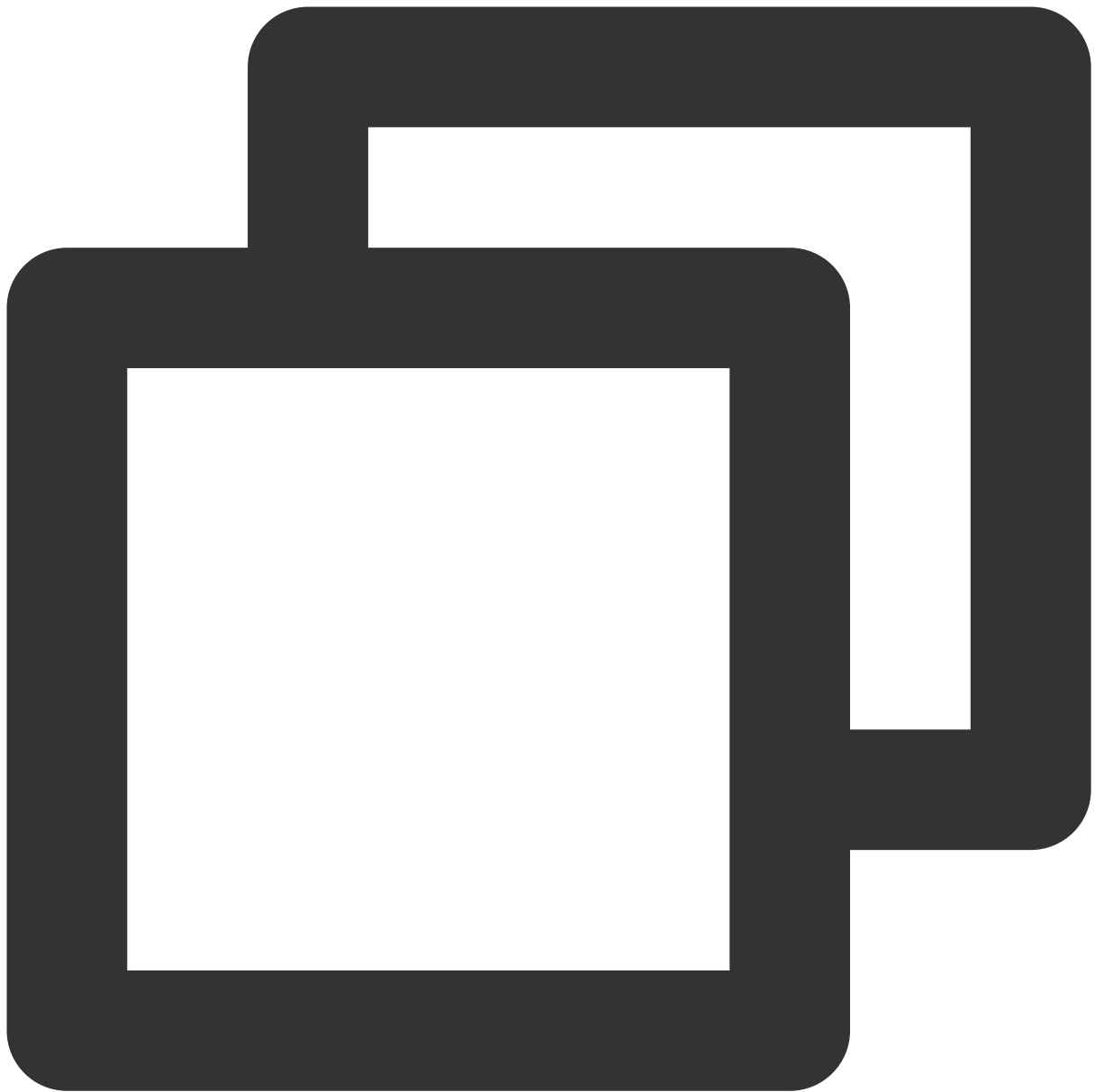
Windows

```
# Run the code in CLI
git clone https://github.com/TencentCloud/chat-uikit-react
# Go to the project
cd chat-uikit-react
# Install dependencies of the demo
npm install && cd examples/sample-chat && npm install
```
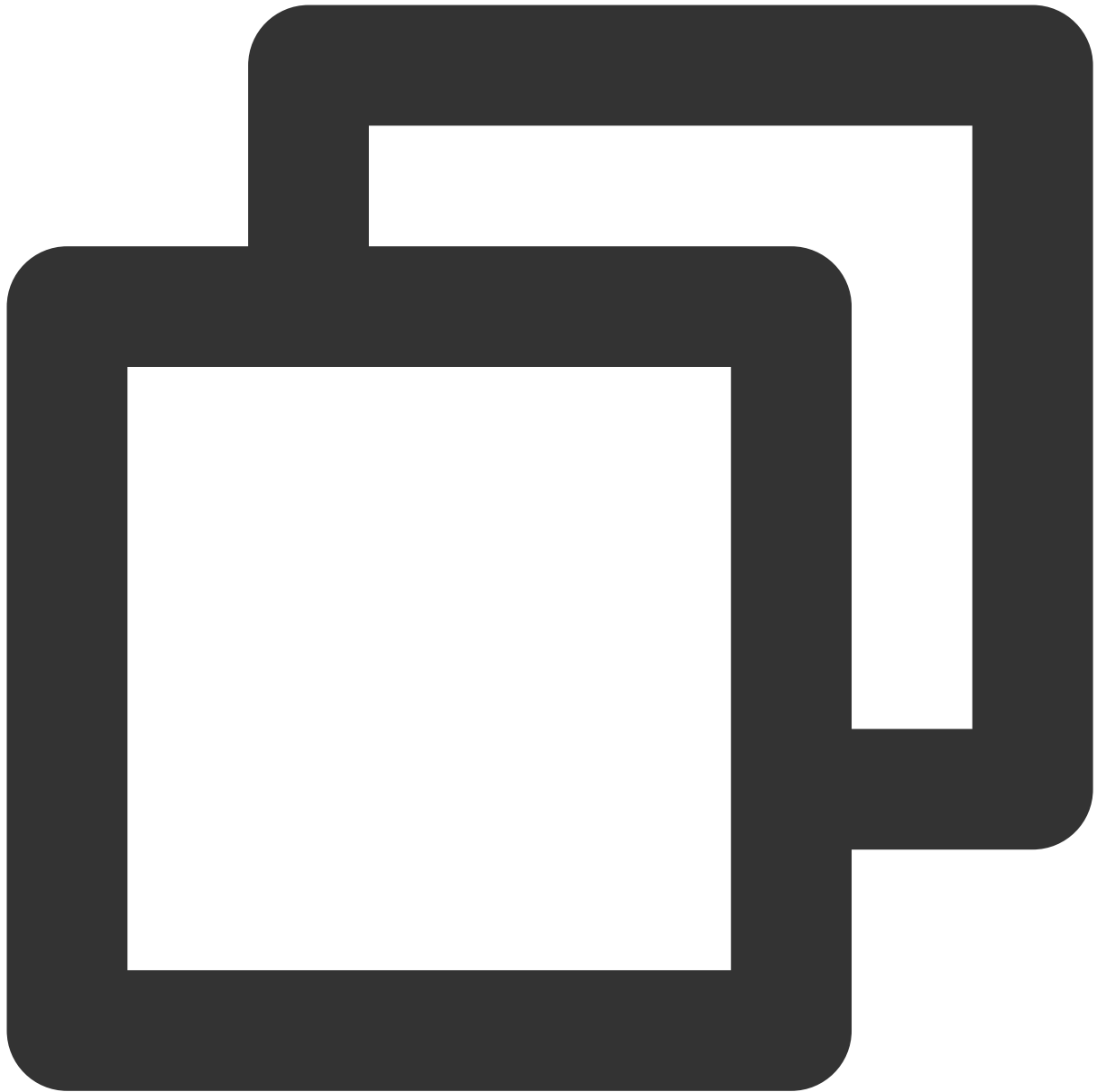
```
# Run the code in CLI
git clone https://github.com/TencentCloud/chat-uikit-react
# Go to the project
cd chat-uikit-react
# Install dependencies of the demo
npm install
cd examples/sample-chat
npm install
```

Step 2: Configure the demo

1. Open the `examples/sample-chat` project and find the `GenerateTestUserSig.js` file in the path `./examples/sample-chat/src/debug/GenerateTestUserSig.js` .

2. In the `GenerateTestUserSig.js` file, set `SDKAPPID` and `SECRETKEY` . Their values can be obtained in the IM console. Click the target application tab to enter its configuration page.

Information such as SDKAppID and secretKey can be obtained in the IM console:

3. Copy the key information and save it to the `./examples/sample-chat/src/debug/GenerateTestUserSig.js` file.

Note:

This document mentions a method for generating UserSig by configuring a secret key in the client code. This method makes the secret key susceptible to decompilation and reverse engineering. Once your key is compromised, attackers can misappropriate your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo and debugging features.**

The correct way to generate UserSig is to integrate the UserSig computation code into your server and provide an API for the app. When UserSig is needed, your app should request a dynamic UserSig from the business server. For more information, see How to Generate a UserSig on the Server.

Step 3: Start the project
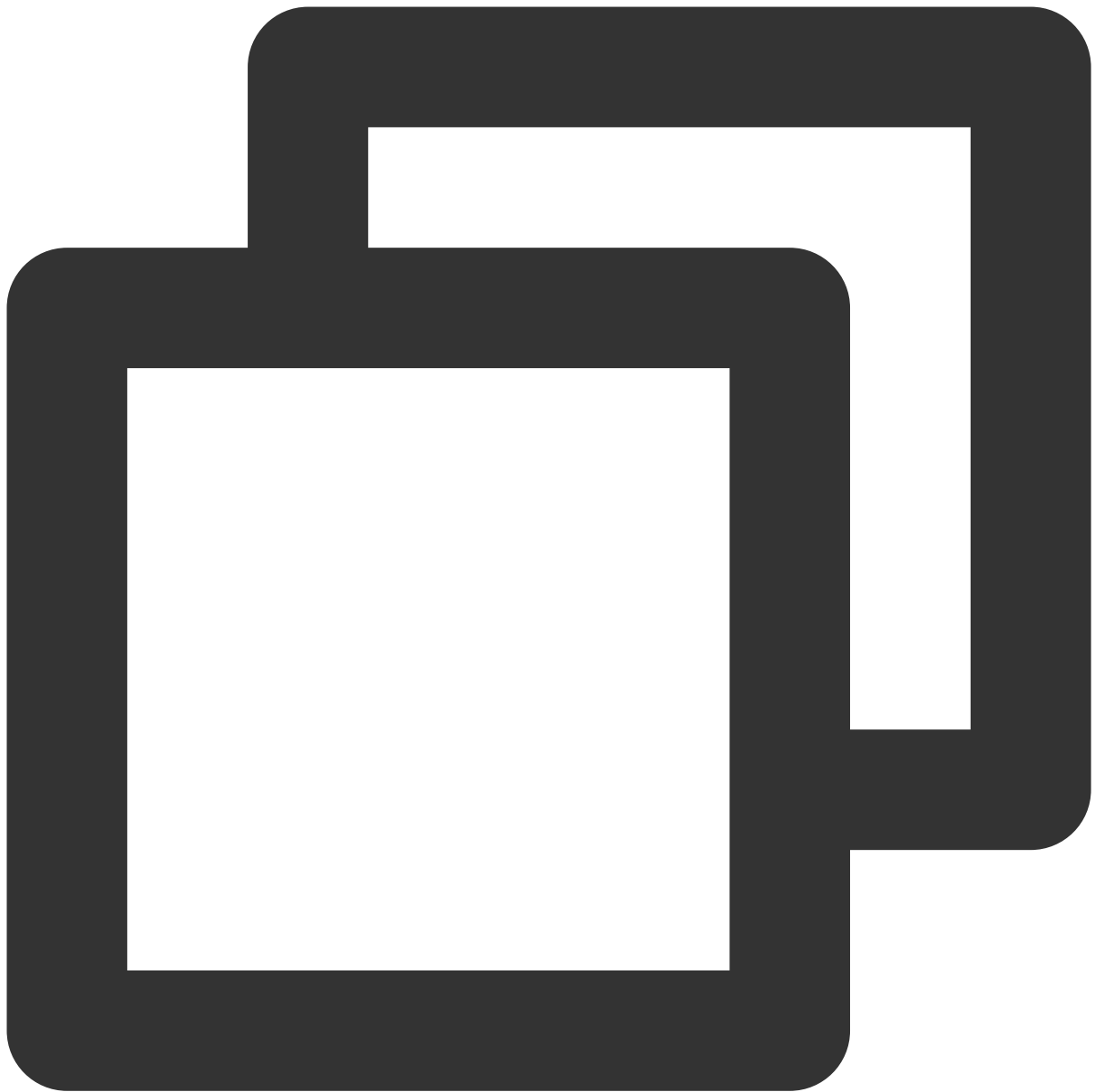
```
# Launch the project
npm run start
```

Step 4: Send your first message

1. After the project is started successfully, click **+** to create a conversation.

2. Enter a userID in the search box.

3. Click the user's avatar to initiate a conversation.

4. Enter a message in the input box and press Enter to send.
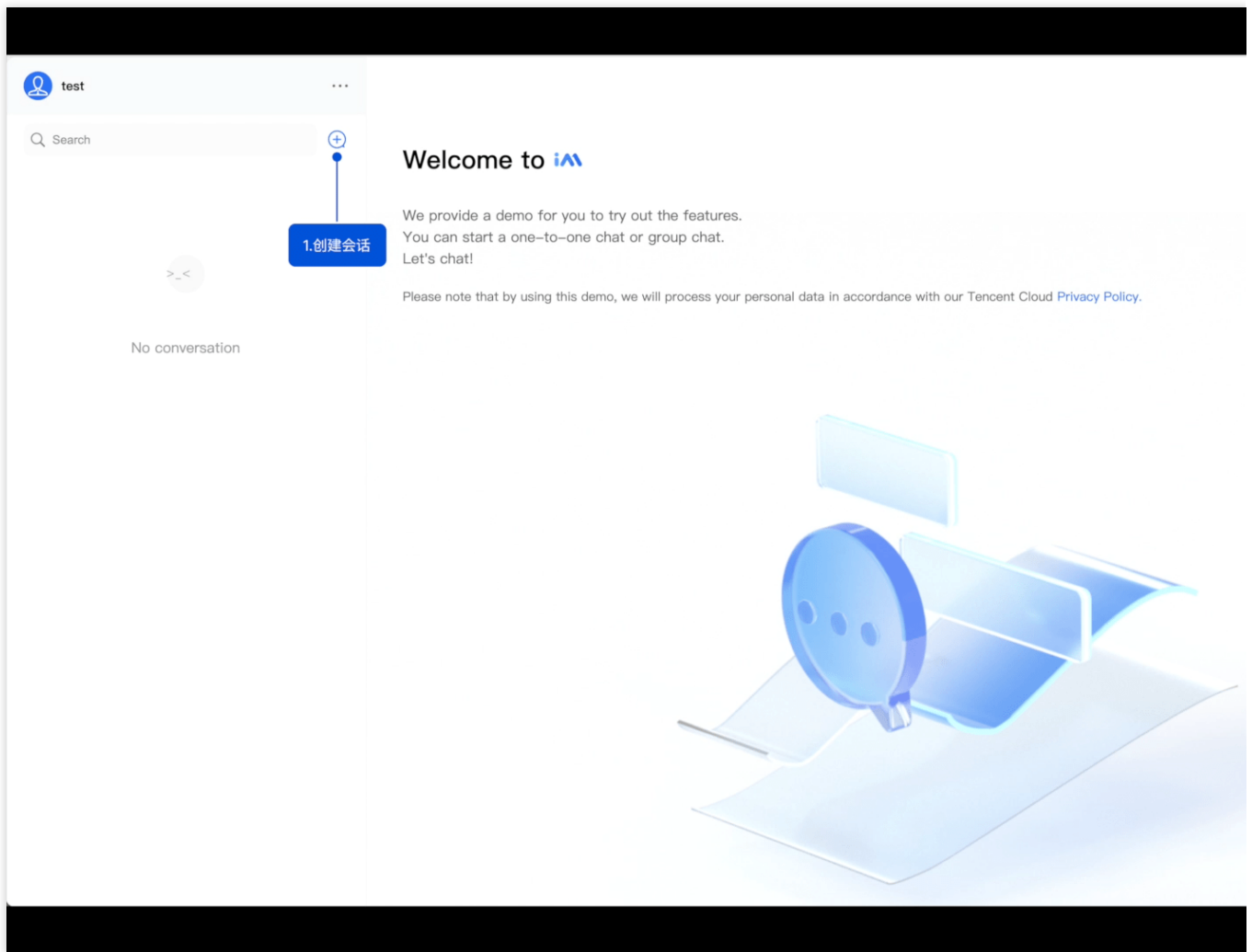
Integrating chat-uikit-react

If you wish to integrate chat-uikit-react into your project, please go to Integration (UI Included)/Integrating Basic Features (React) .

Contact Us

Join the Telegram technical discussion group or WhatsApp discussion group, enjoy the support of professional engineers, and solve your difficulties.

Electron

Last updated：2024-03-15 18:13:48

This document describes how to quickly run the Tencent Cloud Chat demo for Electron and integrate the Electron SDK.

Environment Requirements

Platform	Version
Electron	v13.1.5 or later
Node.js	v14.2.0

Supported Platforms

Currently, both macOS and Windows platforms are supported.

Trying Out Demo

Before integration, you can try out our demo to quickly understand the capabilities of the Tencent Cloud Chat SDK for Electron.

Prerequisites

You have signed up for a Tencent Cloud account and completed identity verification.

Directions

Step 1. Create an app

1. Log in to the Chat console.
Note:

Tencent Cloud

If you already have an app, record its SDKAppID and obtain key information.

A Tencent Cloud account can create a maximum of 300 Chat apps. If you want to create a new app, disable and delete an unwanted app first. **Once an app (along with its SDKAppID) is deleted, the service it provides and all its data are lost. Proceed with caution**.

2. Click **Create Application**, enter your app name, and click **Confirm**.

3. After creation, you can see the status, service version, SDKAppID, tag, creation time, and expiry time of the new app on the overview page of the console. Record the SDKAppID.

Tencent Cloud

4. Click the created app. In the left sidebar, click **Auxiliary Tools** > **UserSig Tools** to create a UserID and the corresponding UserSig. Then copy the UserSig for future login.
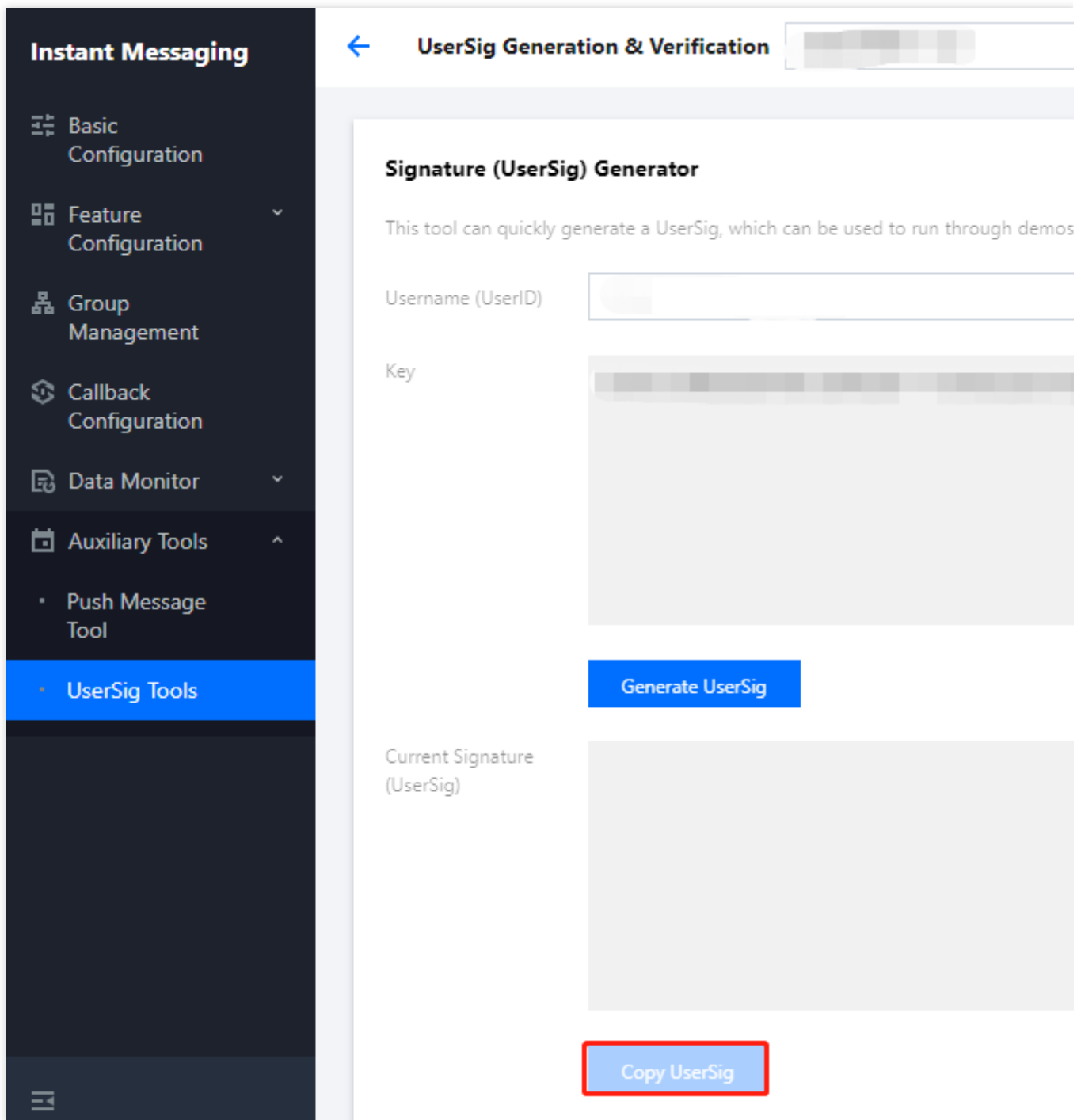
Step 2. Select an appropriate method to integrate the Electron SDK

Tencent Cloud Chat offers two integration schemes:

Integration Scheme	Applicable Scenario
Using a demo	The Chat demo includes all chat features and provides open-source code. If you need to implement chat scenarios, you can use the demo for secondary development. Try it out here.

Self implementation	Implement Chat on your own if the demo does not meet your UI requirements.

To help you better understand Chat SDK APIs, sample APIs are provided here.

Step 3. Use the demo

1. Clone the source code of the Chat Electron demo to the local system.
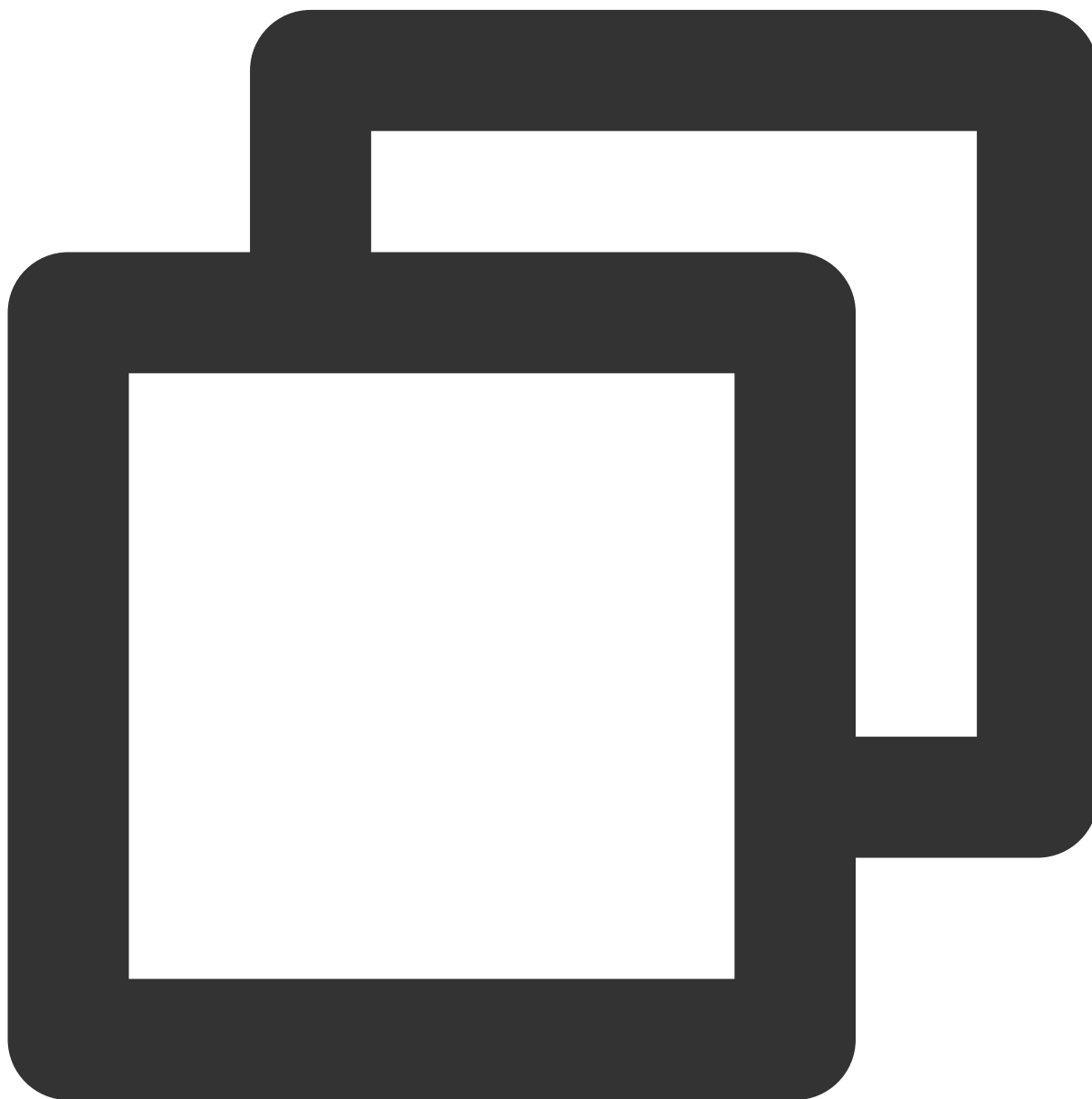
```
git clone https://github.com/TencentCloud/tc-chat-demo-electron.git
```

2. Install project dependencies.
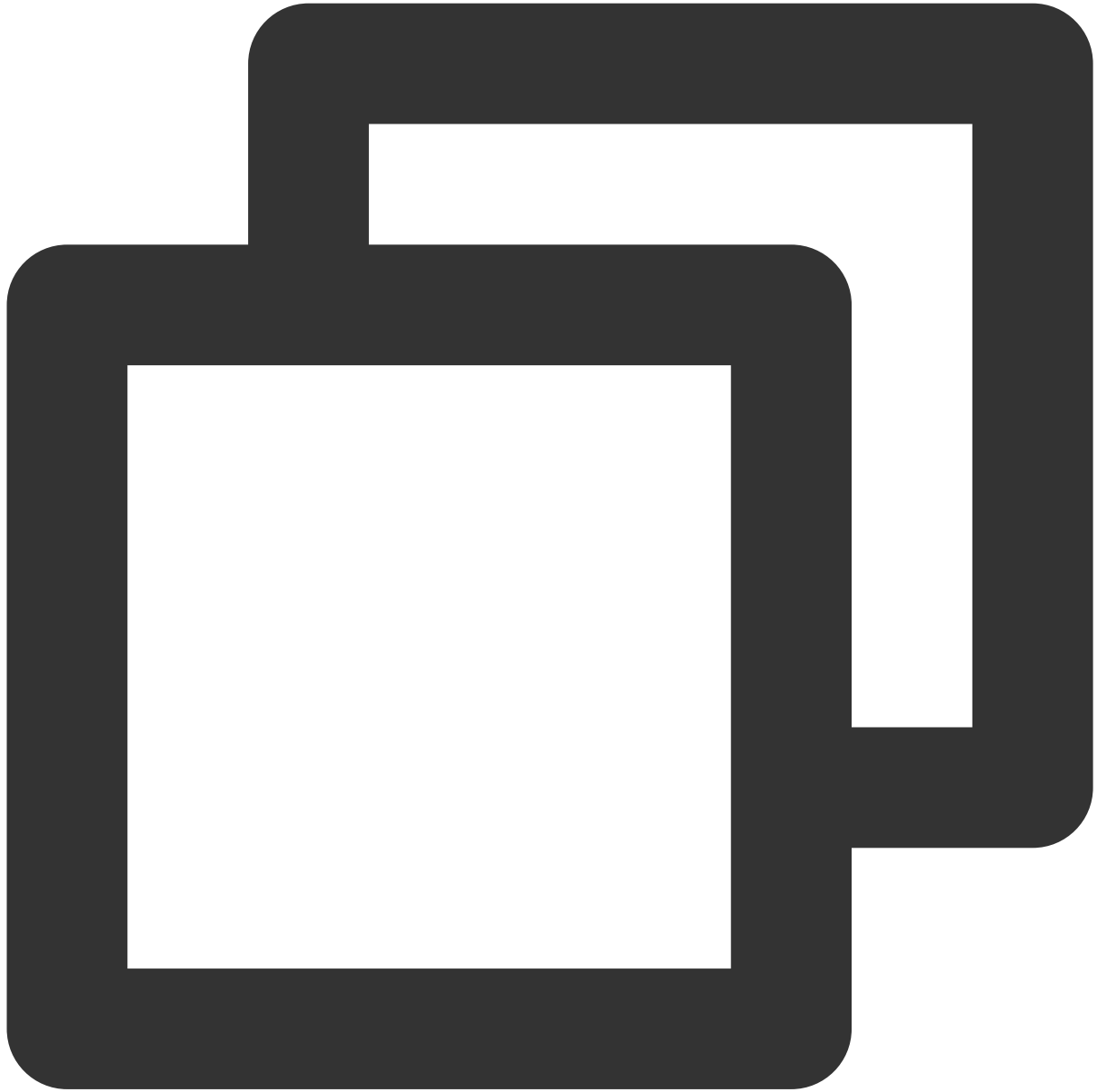
```
// Root directory of the project
npm install

// Rendering process directory
cd src/client
npm install
```

Tencent Cloud
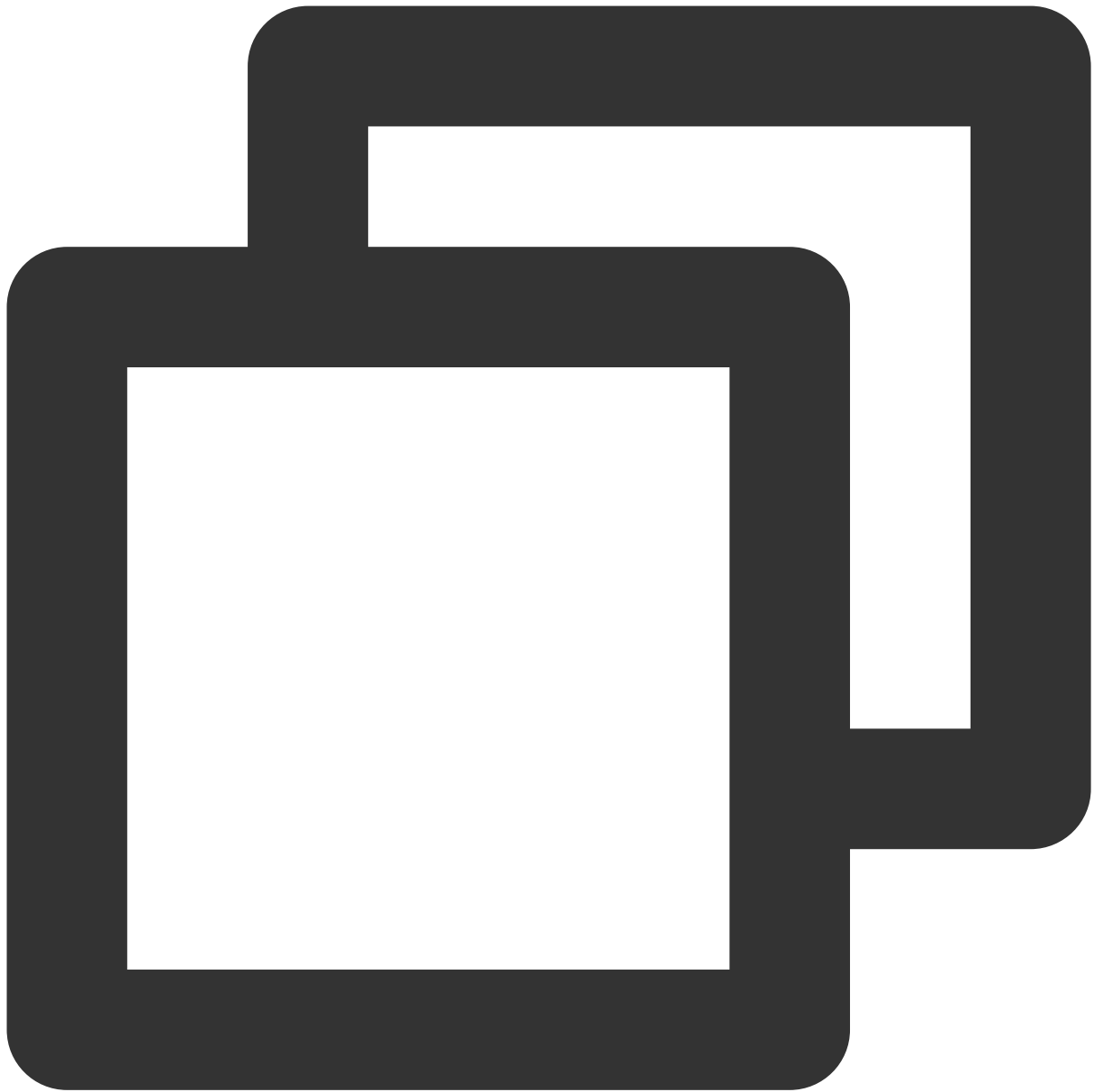
3. Run the project.


```
// Root directory of the project
npm start
```

4. Build the project.

Tencent Cloud
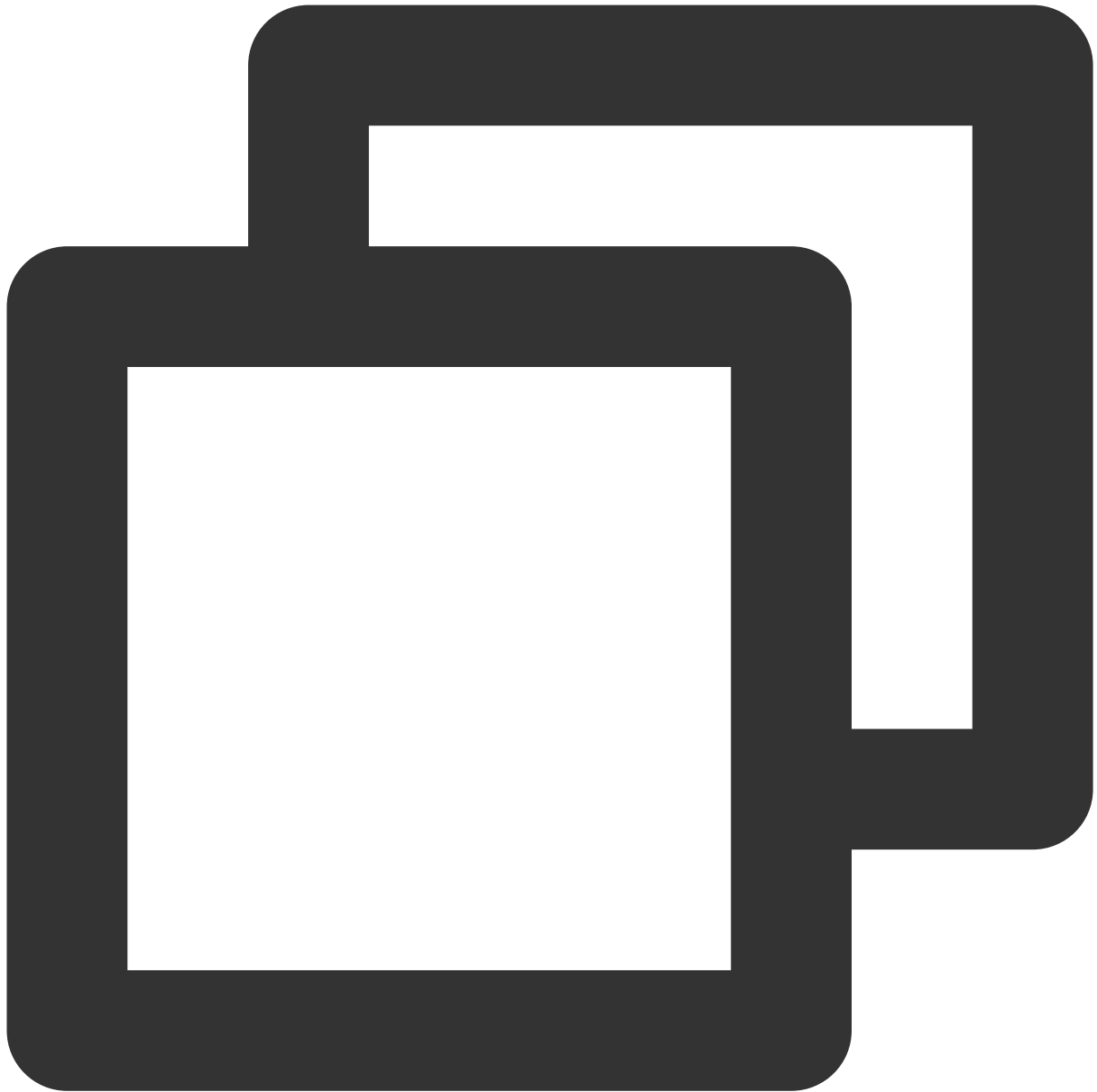
```
// Build the project in macOS
npm run build:mac
// Build the project in Windows
npm run build:windows
```

Note:

In the demo, the main process directory `src/app/main.js` , and the rendering process directory is

`src/client` . If any problem occurs during running, see the FAQs for troubleshooting first.

Step 4. Self implementation

Installing the Electron SDK

Install the latest version of the Electron SDK as follows.

Run the following command:
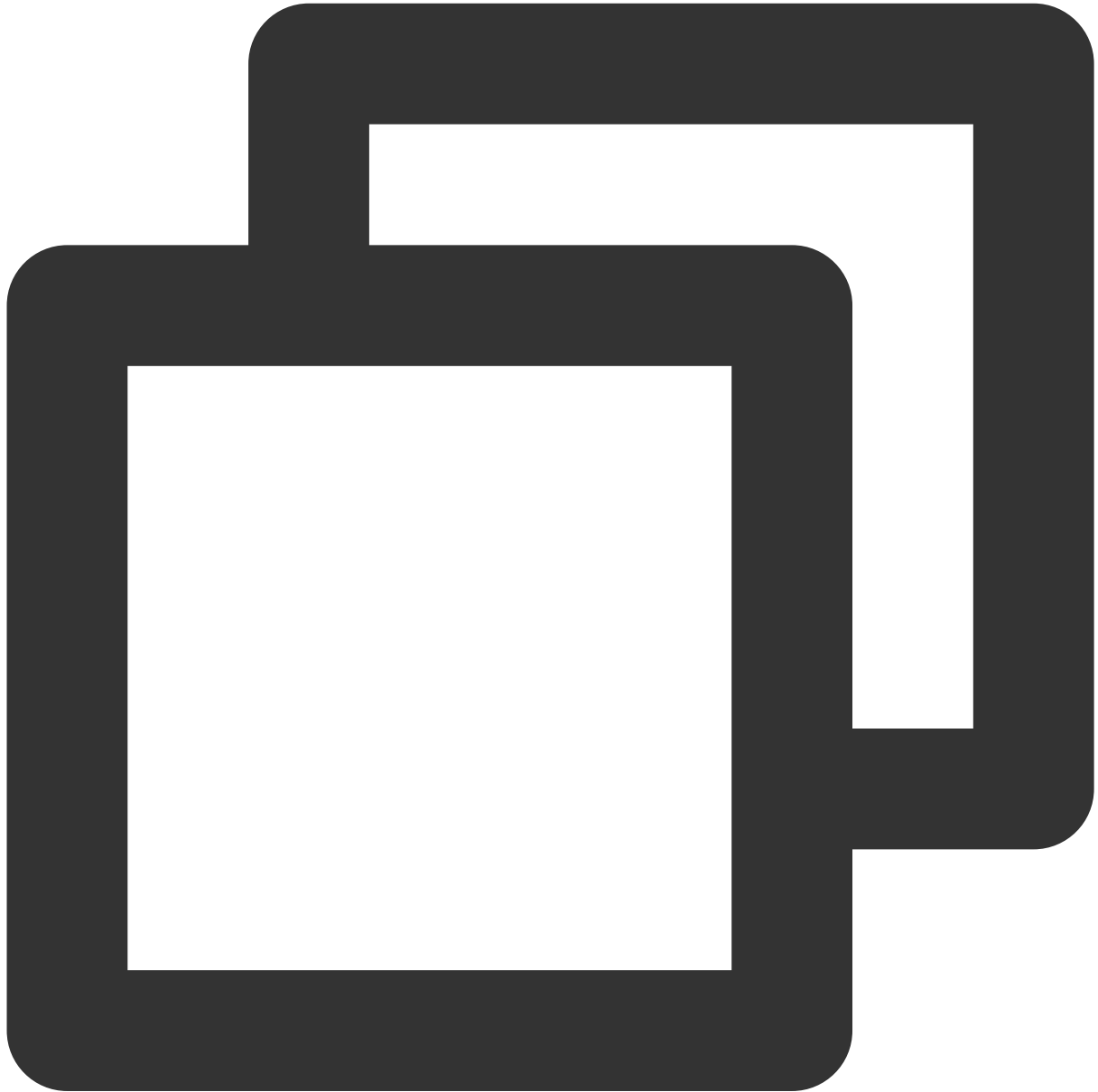
```
npm install im_electron_sdk
```

** Initializing the SDK**

1. Pass in your `sdkAppID` in `TimMain` .
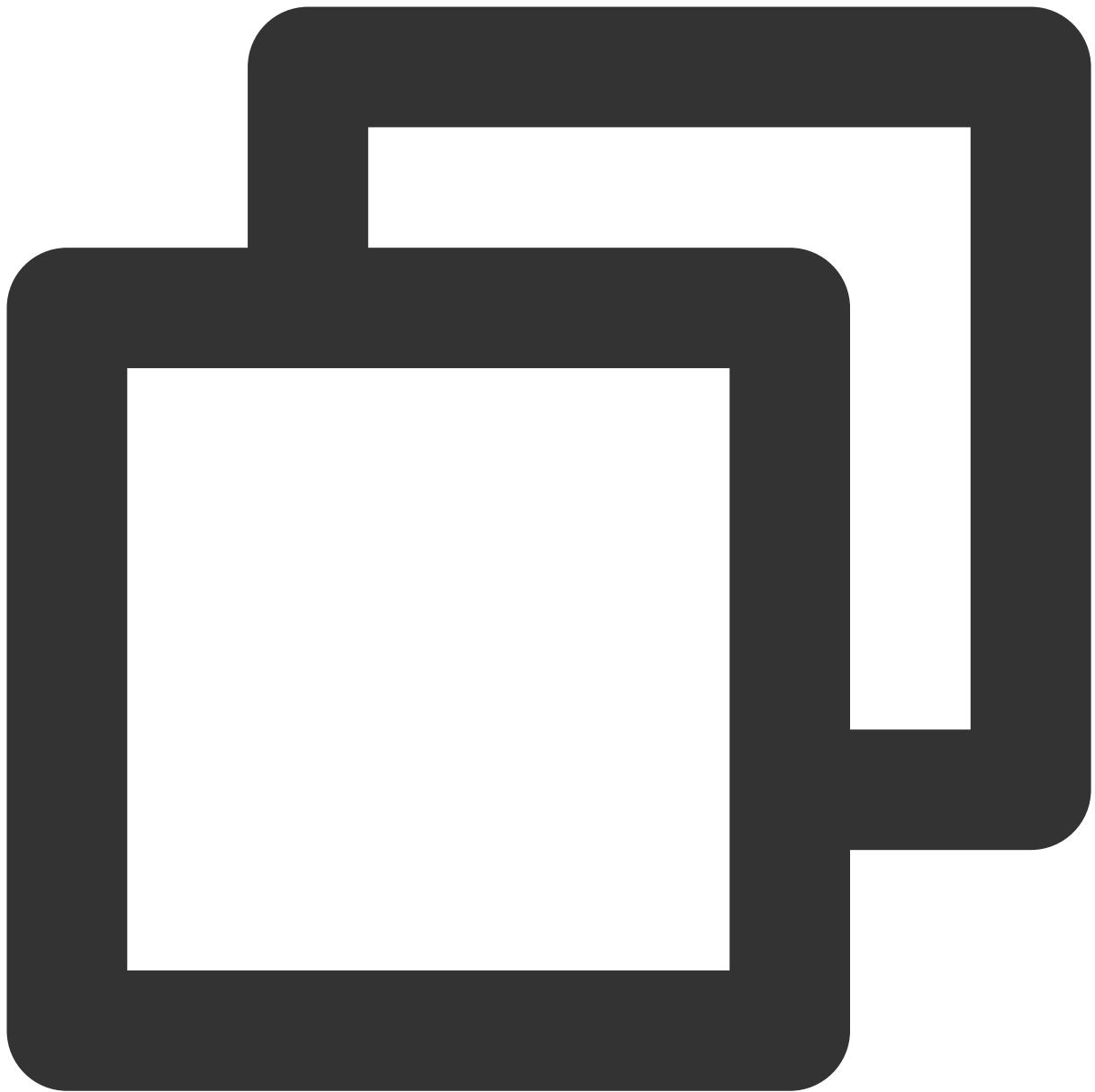
```
// Main process
const TimMain = require('im_electron_sdk/dist/main')

const sdkappid = 0;// You can apply for it in the Chat console.
const tim = new TimMain({
  sdkappid:sdkappid
})
```

2. Call `TIMInit` to initialize the SDK.

```
// Rendering process
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
// Initialize the component
timRender.TIMInit()
```

3. Log in as a test user.

Log in with the test account initially generated in the console for login verification.

Call the `timRender.TIMLogin` method to log in as the test user.

If the returned `code` is `0` , the login is successful.

```
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
let {code} = await timRender.TIMLogin({
  userID:"userID",
  userSig:"userSig" // See how to generate a userSig
})
```

Note:

 This account is for development and testing only. Before the application is launched, the correct `UserSig`

distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-

oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see Generating UserSig.

Sending a message

The following sample shows how to send a text message. If the returned `code` is `0`, the message is sent successfully.

Sample code:
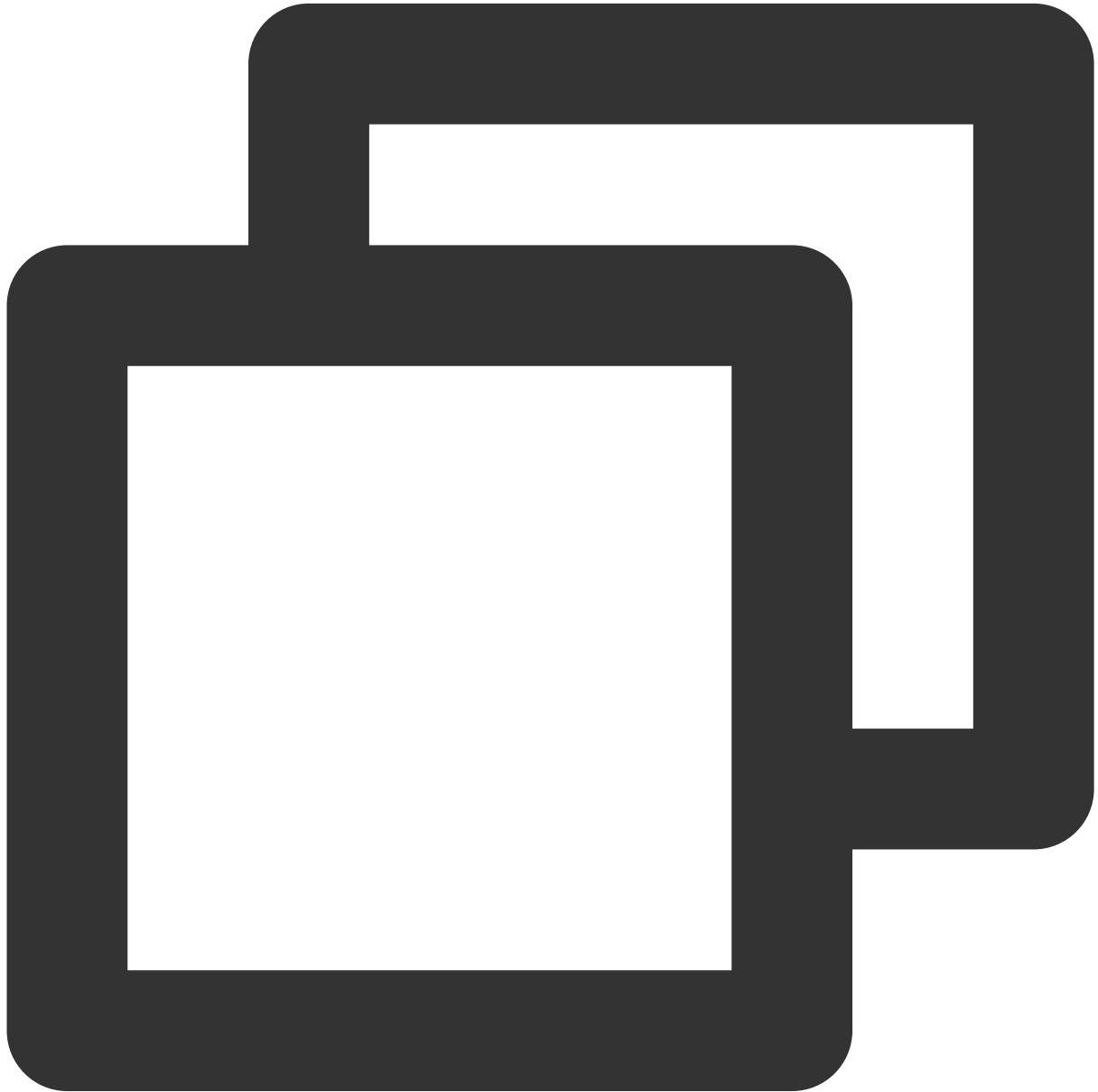
```
const TimRender = require('im_electron_sdk/dist/render')
const timRender = new TimRender();
let param:MsgSendMessageParamsV2 = {    // param of TIMMsgSendMessage
```

```
    conv_id: "conv_id",
    conv_type: 1,
    params: {
        message_elem_array: [{
            elem_type: 1,
            text_elem_content:'Hello Tencent!',

        }],
        message_sender: "senderID",
    },
    callback: (data) => {}
  }
let {code} = await timRender.TIMMsgSendMessageV2(param);
```

Note:

If sending the message fails, it may be that your `sdkAppID` does not support sending messages to strangers. In this case, you can disable the friend relationship chain check in the console for testing.

Getting the conversation list

Log in with the other test account to pull the conversation list.

Common use cases include:

Get the conversation list upon application start and listen for the persistent connection to update the conversation list in real time.
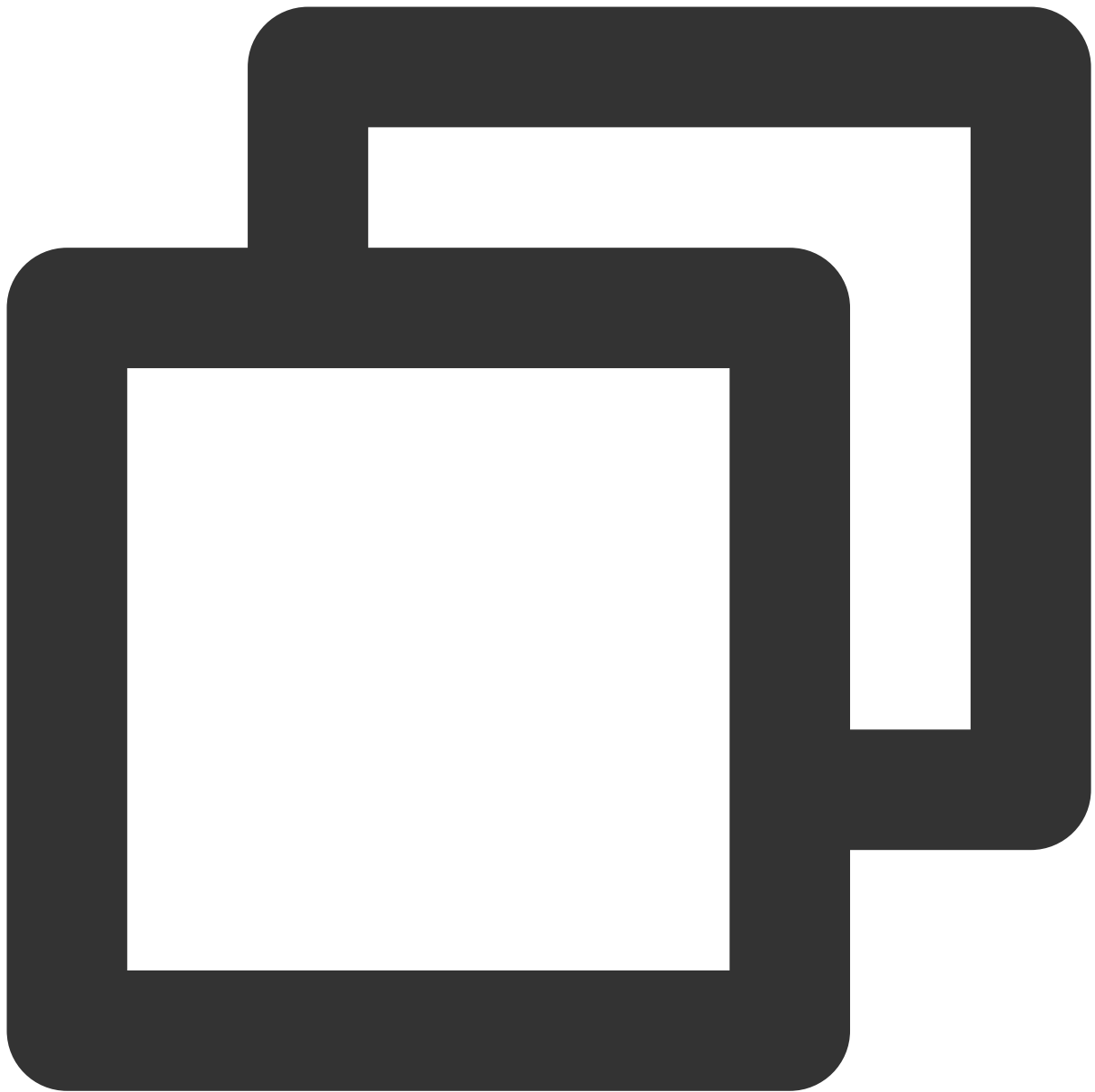
```
let param:getConvList = {
        userData:userData,
    }
let data:commonResult<convInfo[]> = await timRenderInstance.TIMConvGetConvList(para
```

At this point, you can see the message sent by the other test account in the previous step.

Receiving a message

Common use cases include:

1. After a new conversation is opened on the UI, request a certain number of historical messages at a time to display the historical message list.

Tencent Cloud

2. Listen for the persistent connection to receive messages in real time and add them to the historical message list.

Requesting the historical message list at a time
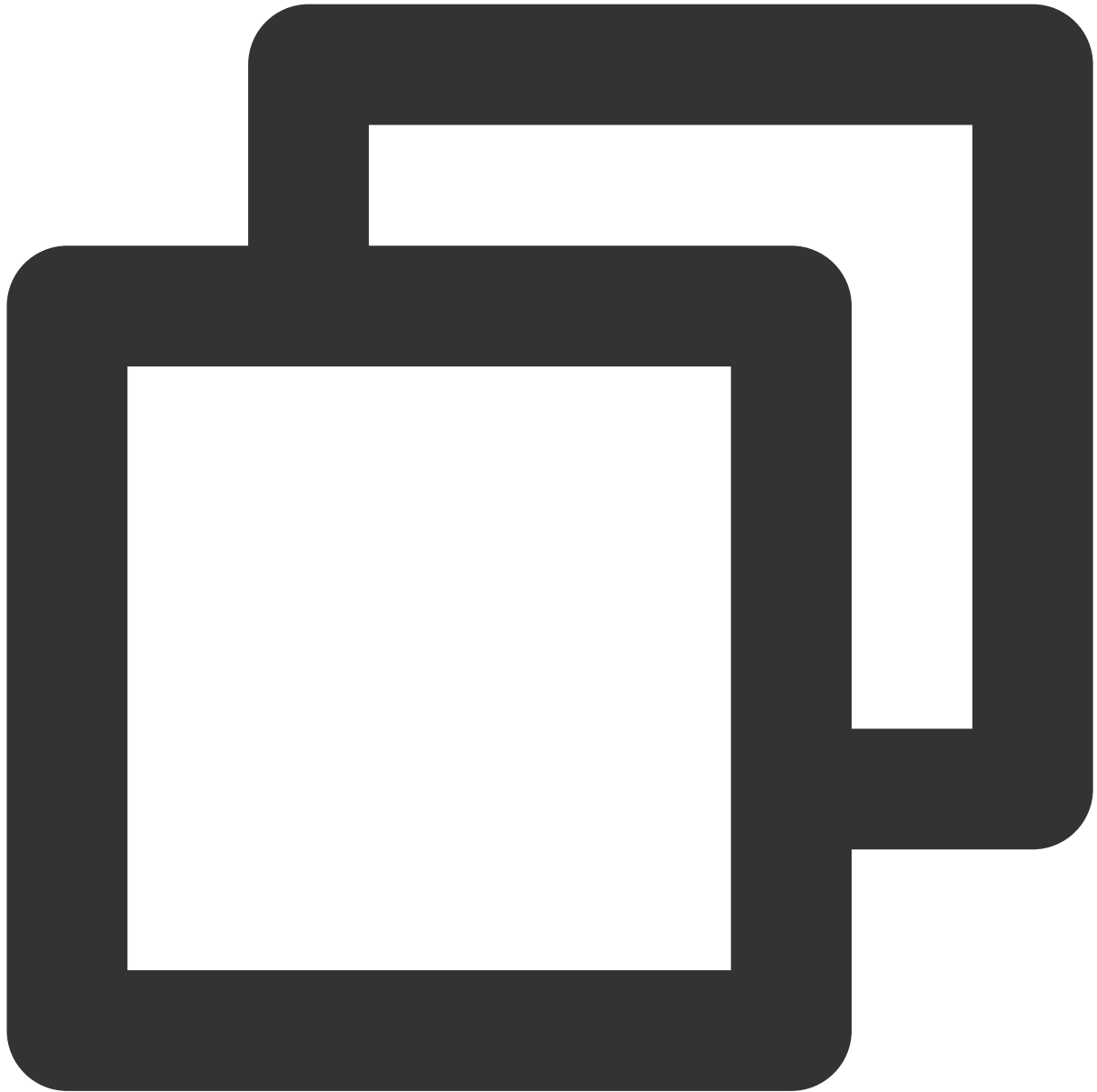

```
let param:MsgGetMsgListParams = {
        conv_id: conv_id,
        conv_type: conv_type,
        params: {
            msg_getmsglist_param_last_msg: msg,
            msg_getmsglist_param_count: 20,
            msg_getmsglist_param_is_remble: true,
        },
```

Tencent Cloud

```
        user_data: user_data
    }
    let msgList:commonResult<Json_value_msg[]> = await timRenderInstance.TIMMsgGetM
```

Listening for new messages in real time

The following is the sample code for callback binding:
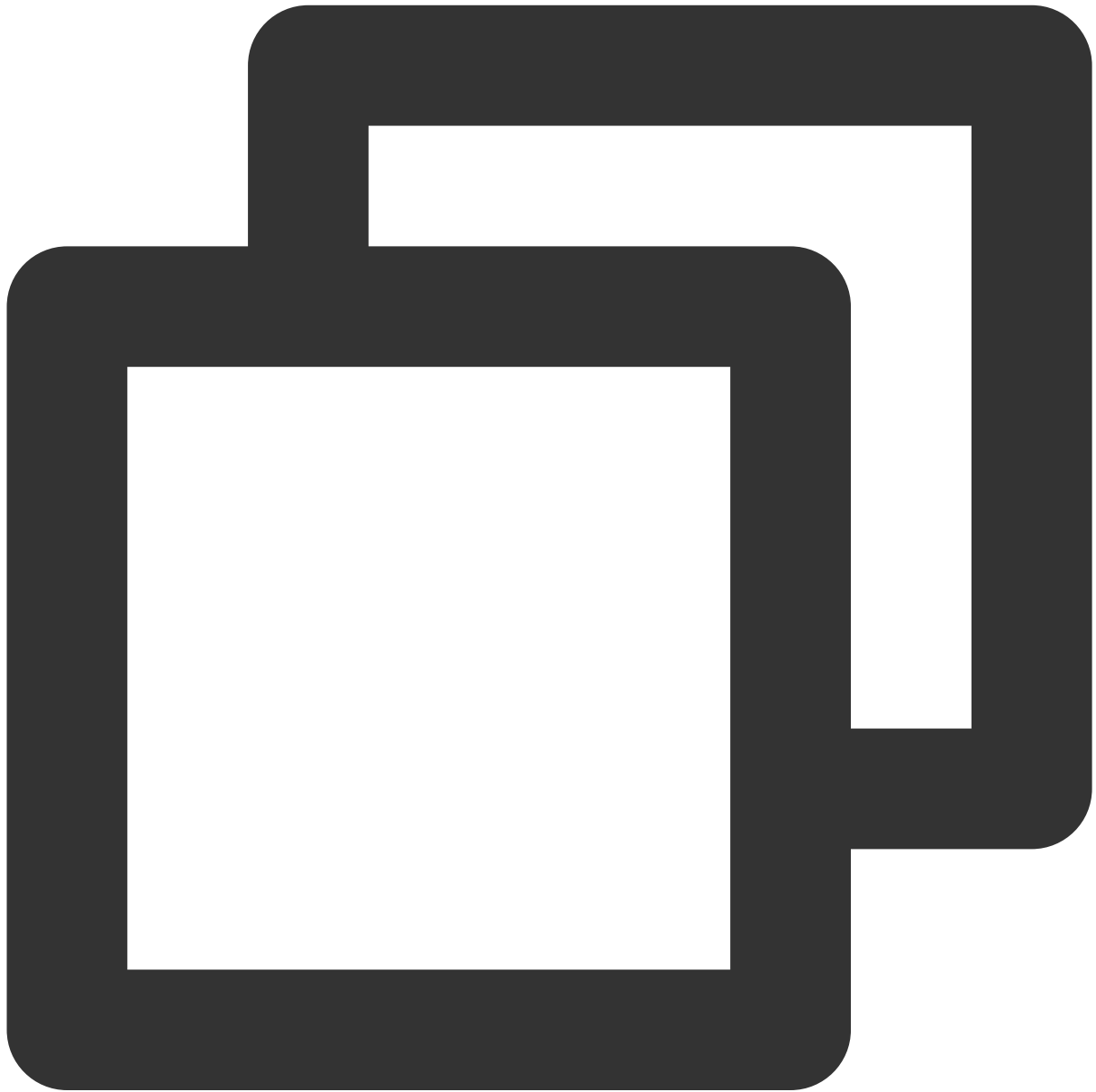
```
let param : TIMRecvNewMsgCallbackParams = {
        callback: (...args)=>{},
        user_data: user_data
    }
```

```
timRenderInstance.TIMAddRecvNewMsgCallback(param);
```

At this point, you have completed the Chat module development, and now users can send and receive messages and enter different conversations.

You can develop more features, such as group, user profile, relationship chain, offline push, and local search.

For detailed directions, see here.

FAQs

What platforms are supported?

Currently, both macOS and Windows platforms are supported.

How do I query error codes?

For Chat SDK API error codes, see Error Codes.

What should I do if the error `npm ERR! gyp ERR! stack TypeError: Cannot assign to read only property 'cflags' of object '#<Object>'` **is reported during development environment installation?**

Downgrade the node version to 16.18.1.

What should I do if the error `gypgyp ERR!ERR` **is reported during development environment installation?**

See gypgyp ERR!ERR! .

What should I do if the error `npm ERR! Fix the upstream dependency conflict, or retry` **is reported when** `npm install` **is run?**

In versions earlier than npm v7, dependency conflicts that occur during installation are automatically ignored.

In npmv7 or later versions, dependency conflicts will not be automatically ignored, and you need to manually enter a command to ignore them.

The command for ignoring dependency conflicts is as follows:

Tencent Cloud
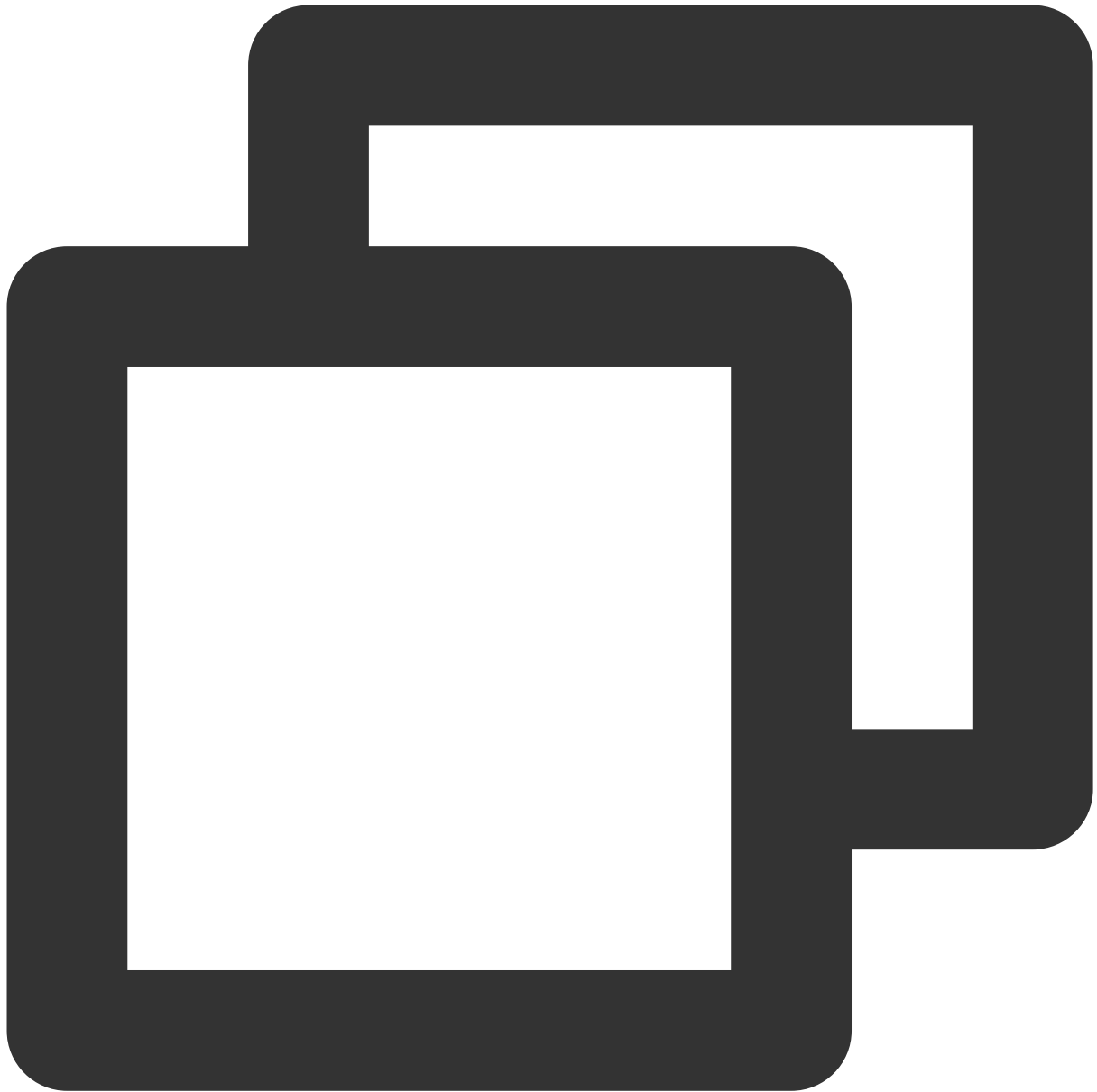

```
npm install --force
```

What should I do if the error `Error: error:0308010C:digital envelope routines::unsupported` **is reported when** `npm run start` **is run?**

Downgrade the node version to 16.18.1.

What should I do if the screen turns white when I run `npm run start` **on a macOS client demo?**

The error occurs because the rendering process code is not completely built and the port 3000 opened by the main process points to an empty page. The error will be resolved after the rendering process code is completely built and you refresh the window. Alternatively, you can run `cd src/client && npm run dev:react` and `npm run dev:electron` to start the rendering process and main process separately.

How do I use native modules in projects built with vue-cli-plugin-electron-builder?

For issues related to using native modules in projects built with vue-cli-plugin-electron-builder, see No native build was found for platform = xxx.

How do I use native modules in projects built with webpack?

For issues related to using native modules in projects built with webpack, see FAQs in the Windows environment.

What should I do if the error "Dynamic Linking Error" is reported?

Dynamic Linking Error. electron-builder configuration
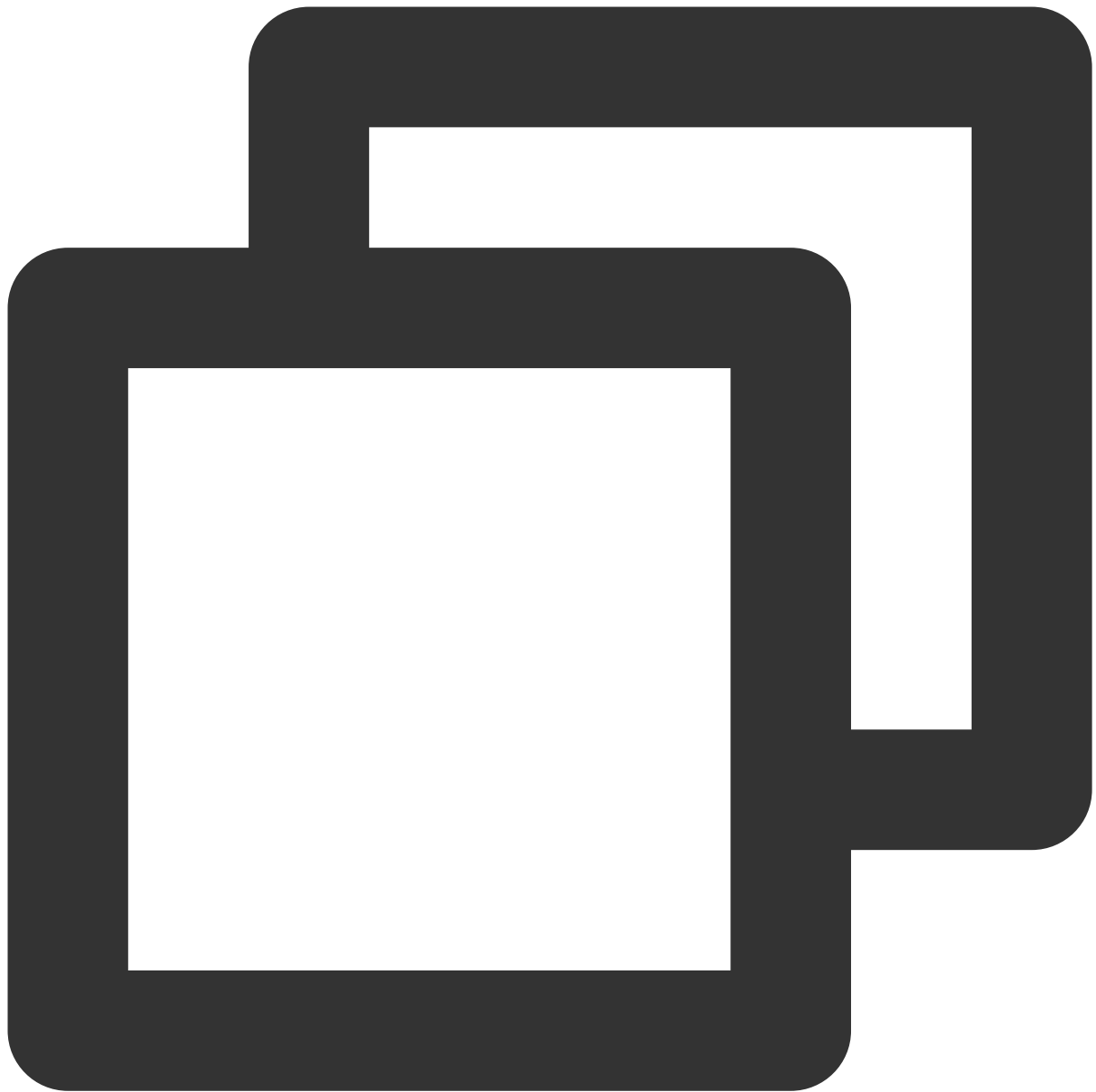
```
extraFiles:[
  {
    "from": "./node_modules/im_electron_sdk/lib/",
    "to": "./Resources",
    "filter": [
      "**/*"
    ]
  }
]
```

Getting `__dirname is not defined` **when using electron-vite?**

Since electron-vite does not support node integration and communicating between main and renderer processes in renderer process, Tencent Cloud Chat SDK needs to be written in `preload` for use. The code for Main process should be written in main process normally. For details, please refer to electron-vite documentation.

The usage is the same. Please refer to the example code of the document. Taking initialization as an example, the example code is as follows:
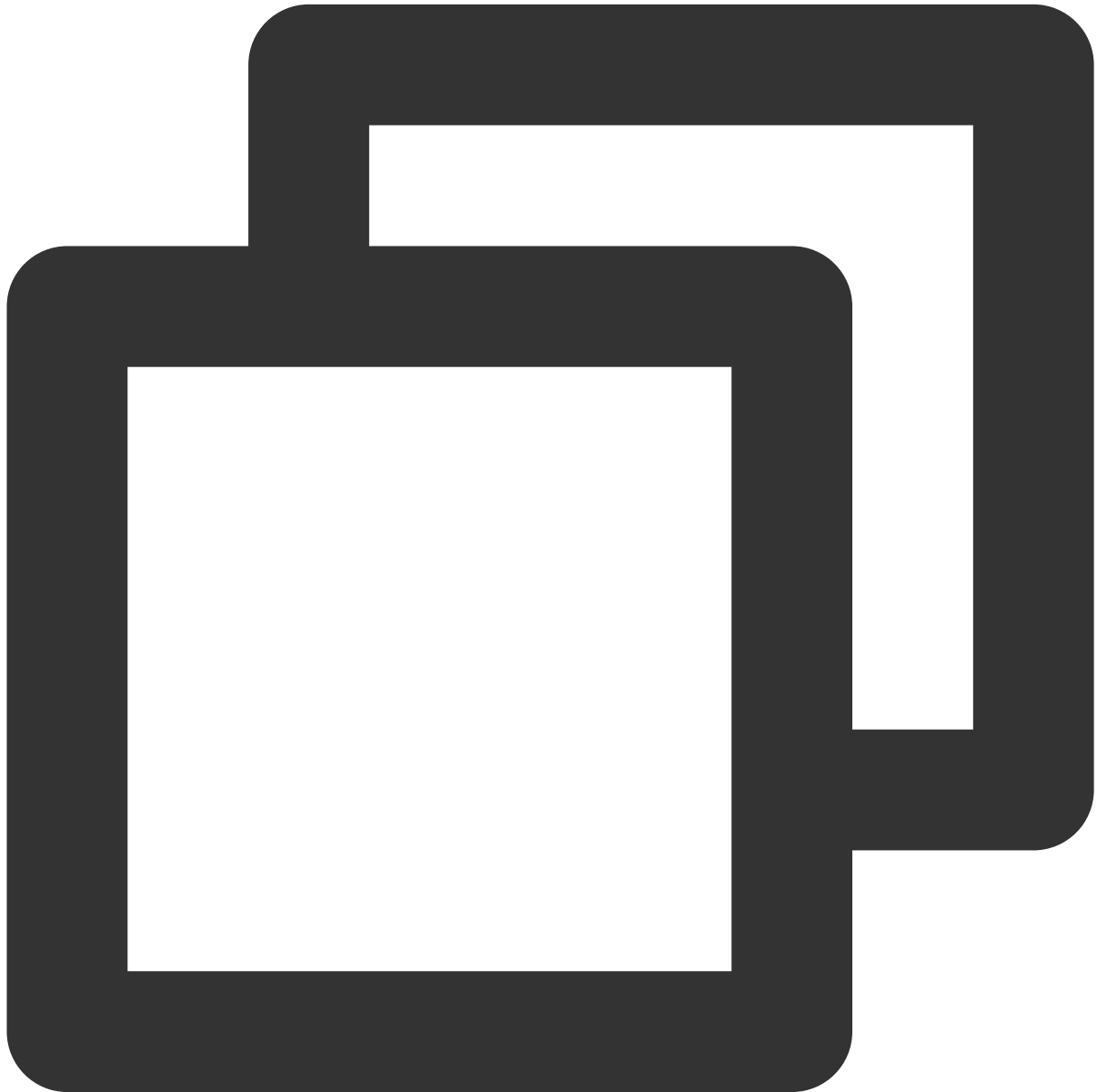
```
//The content of the main process is written to the main process
// main/index.ts (example path)
const TimMain = require('im_electron_sdk/dist/main')
```

```
const sdkappid = 0;
const tim = new TimMain({
    sdkappid:sdkappid
})
// Use chat sdk in preload
// preload/index.ts (example path)
import TimRender from 'im_electron_sdk/dist/renderer'
const timRender = new TimRender();
```

uniapp

Last updated：2024-02-01 11:12:57

Introduction to chat-uikit-uniapp

chat-uikit-uniapp (vue2 /vue3) is a uniapp UI component library based on Tencent Cloud Chat SDK. It provides universally used UI components that include Conversation, Chat, and Group components. Leveraging these meticulously crafted UI components, you can quickly construct an elegant, reliable, and scalable Chat application. The interface of chat-uikit-uniapp is as demonstrated in the image below:
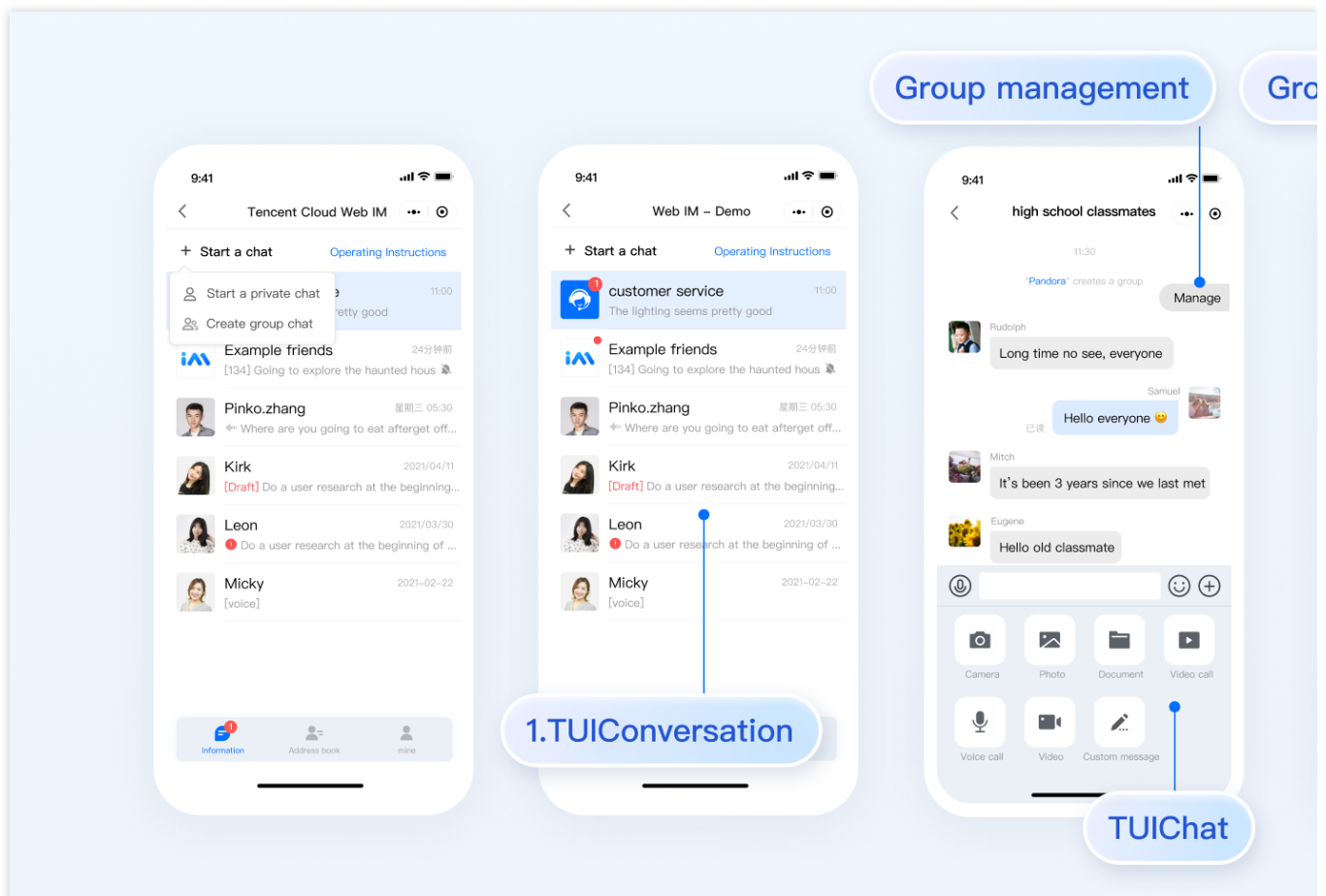
Supported Platform

Android

iOS

WeChat Mini Program

H5

Environment Requirements

HBuilderX (HBuilderX Version >= 3.8.4.20230531) or upgrade to the newest version

Vue2 / Vue3

Sass (sass-loader version ≤ 10.1.1)

Node (12.13.0 ≤ node version ≤ 17.0.0. The official LTS version 16.17.0 of Node.js is recommended.)

npm (use a version that matches the Node version in use)

TUIKit Source Code Integration

Follow the steps below to send your inaugural message.

Step 1: create a project (ignore this step if already has project)

Launch HbuilderX, select "File-New-Project" in the menu bar, and create a uni-app project named `chat-example` .
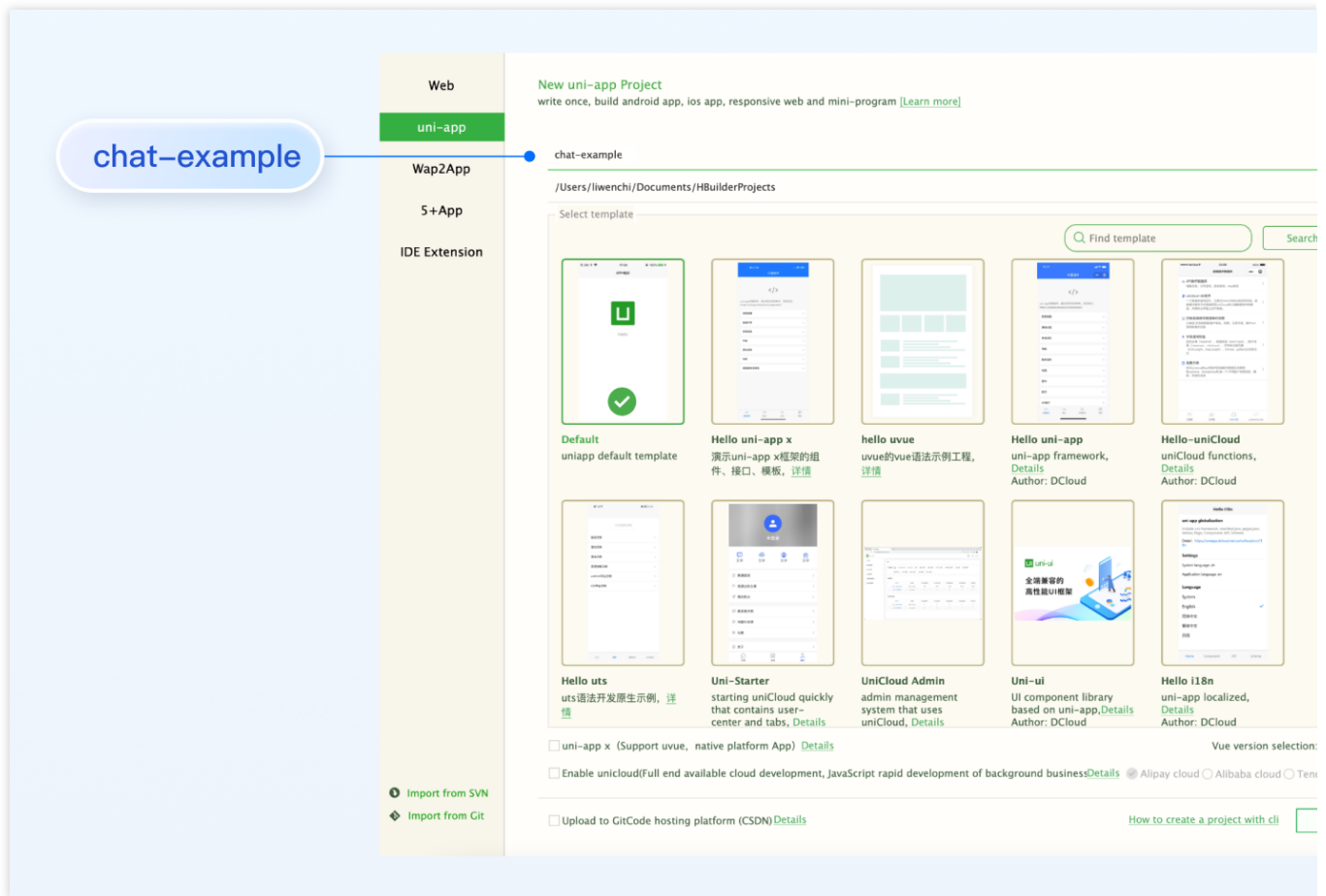
Step 2. Download the TUIKit component

Since HBuilderX does not create package.json files by default, you need to proactively create one. Execute the following command in the root directory of the project:
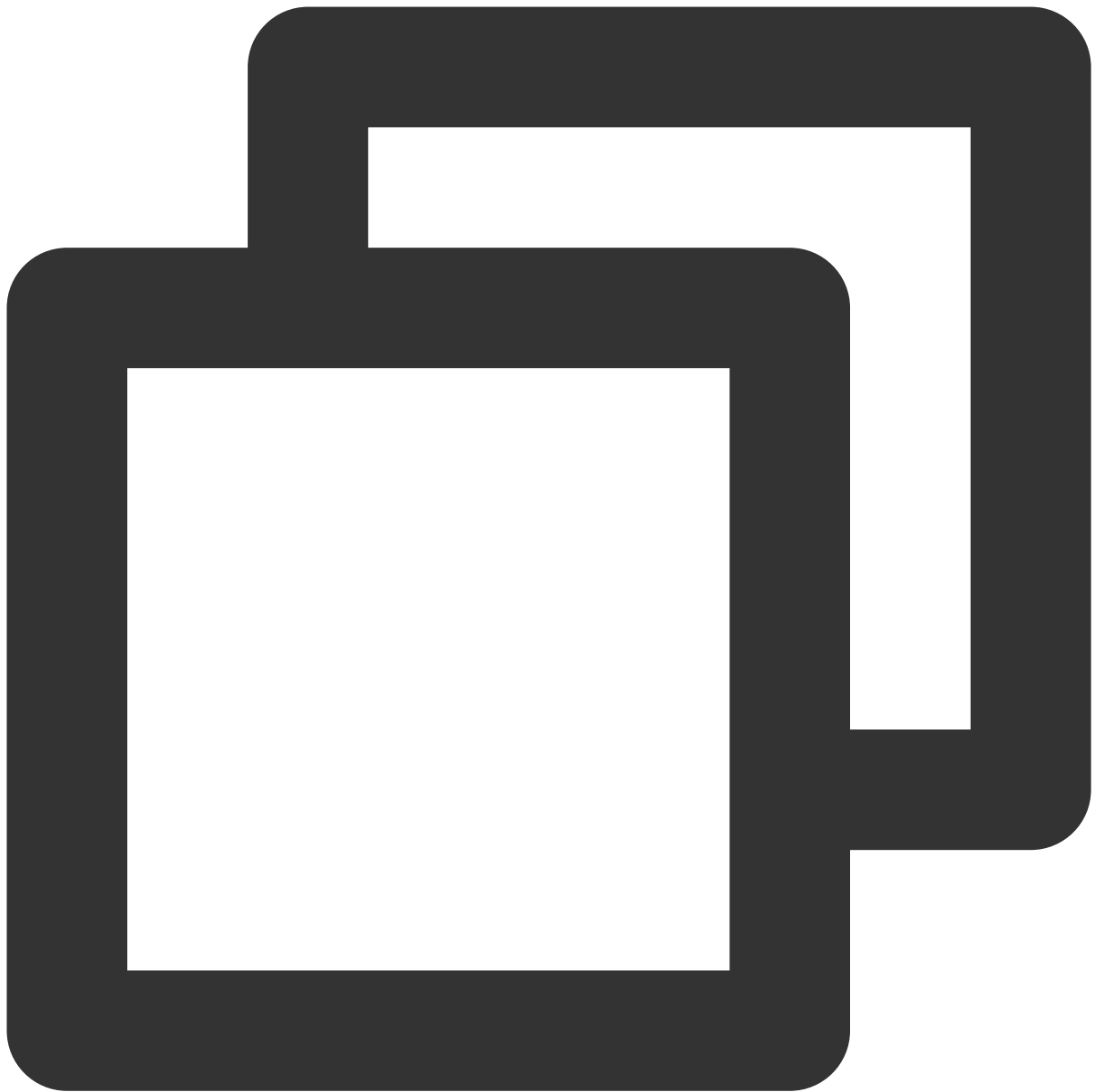
```
npm init -y
```

Download TUIKit and copy it to the source code:

macOS

Windows

Download the TUIKit component using the npm method:
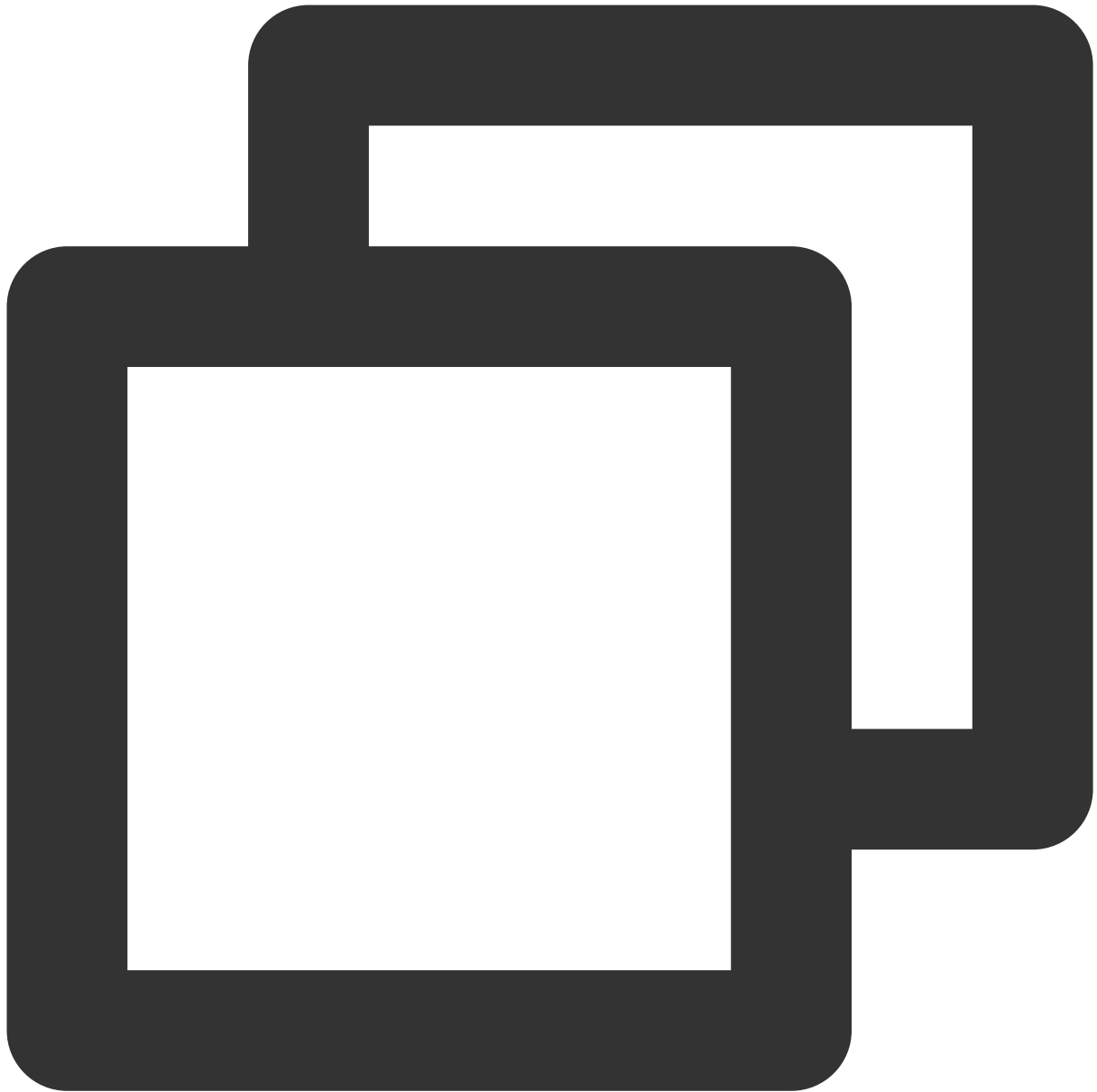
```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

For ease of subsequent extensions, we propose that you replicate the TUIKit component to the pages directory within your project. Please conduct the following command in the root directory of your own project:
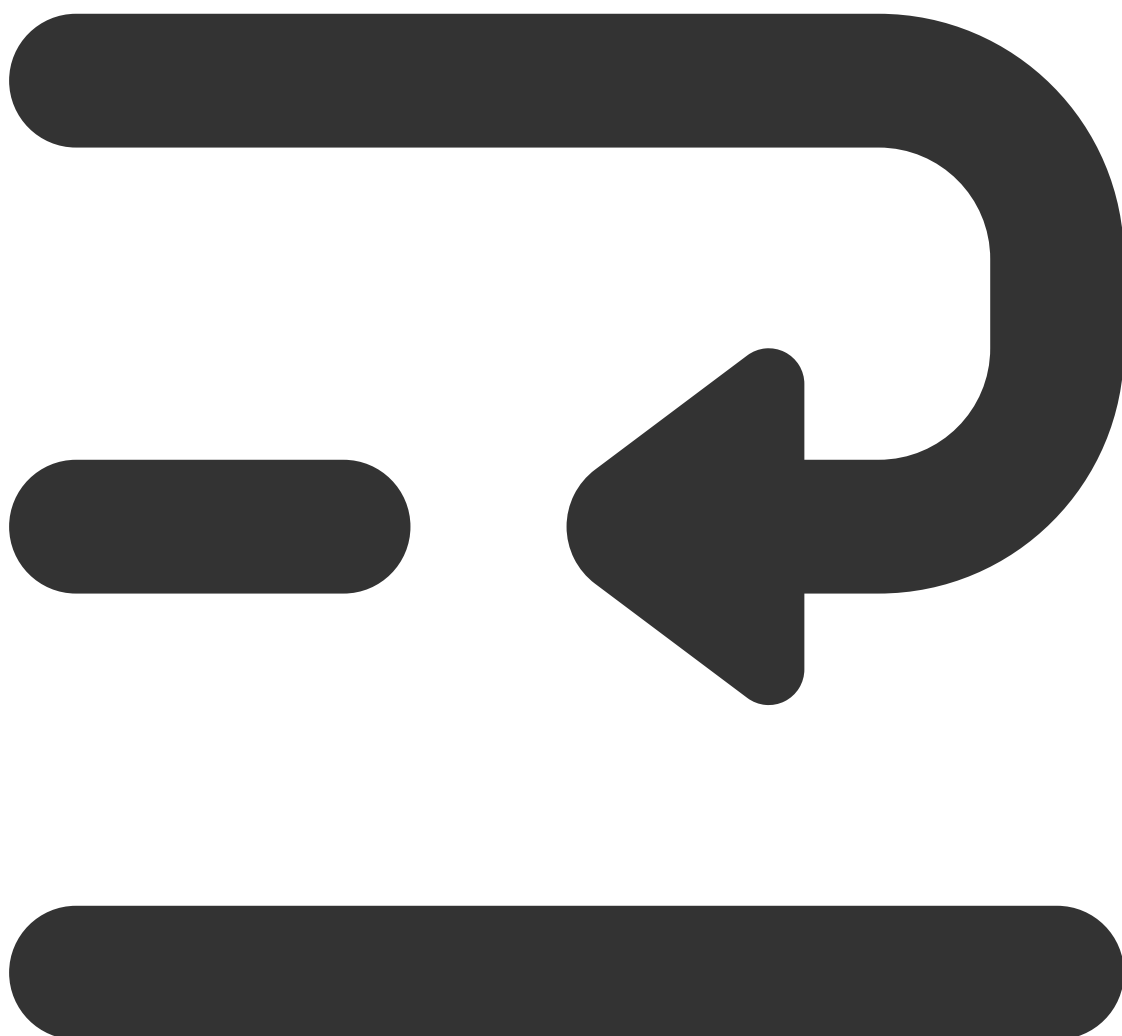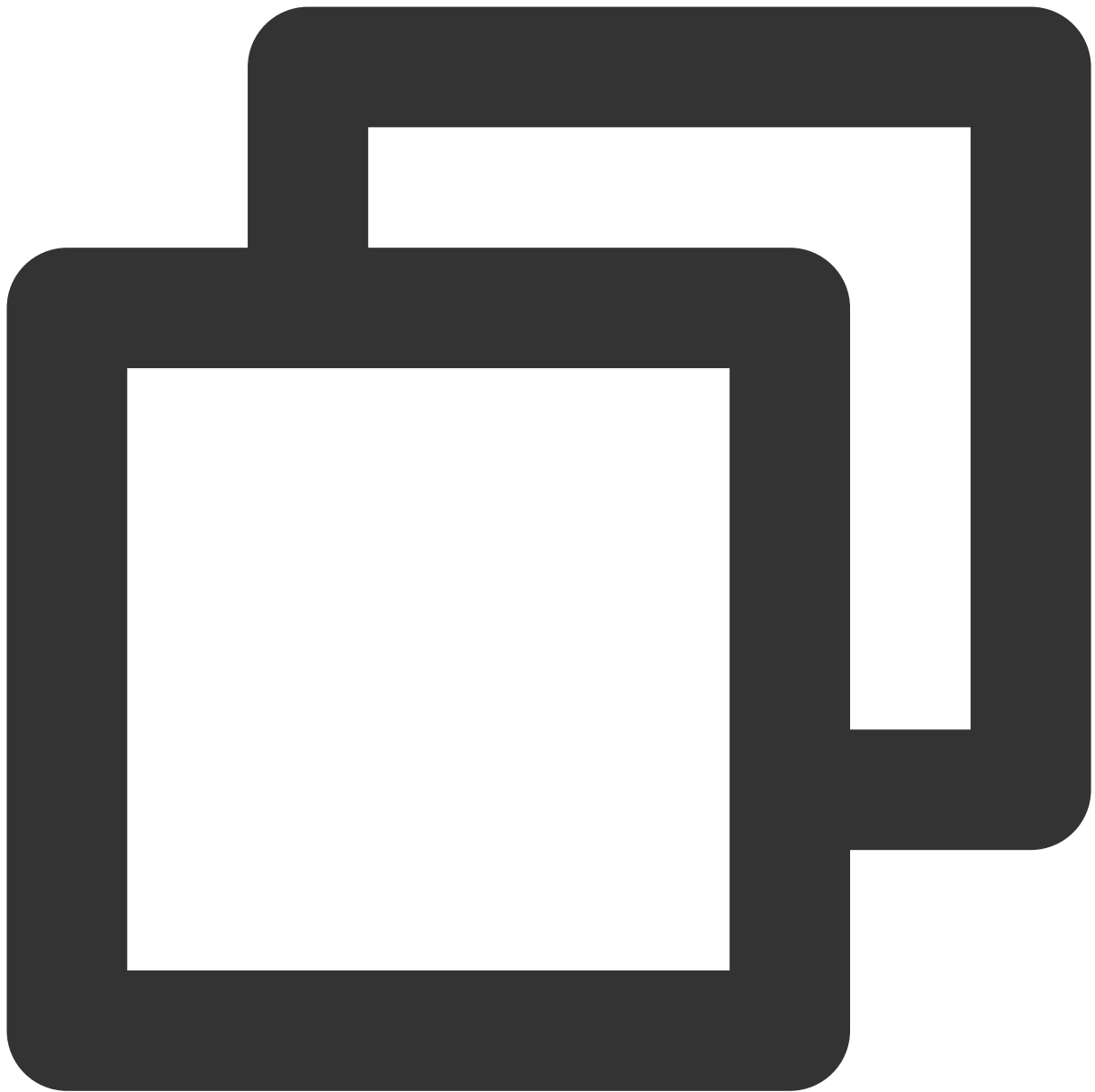
```
mkdir -p ./TUIKit && rsync -av --exclude={'node_modules','package.json','excluded-l
```
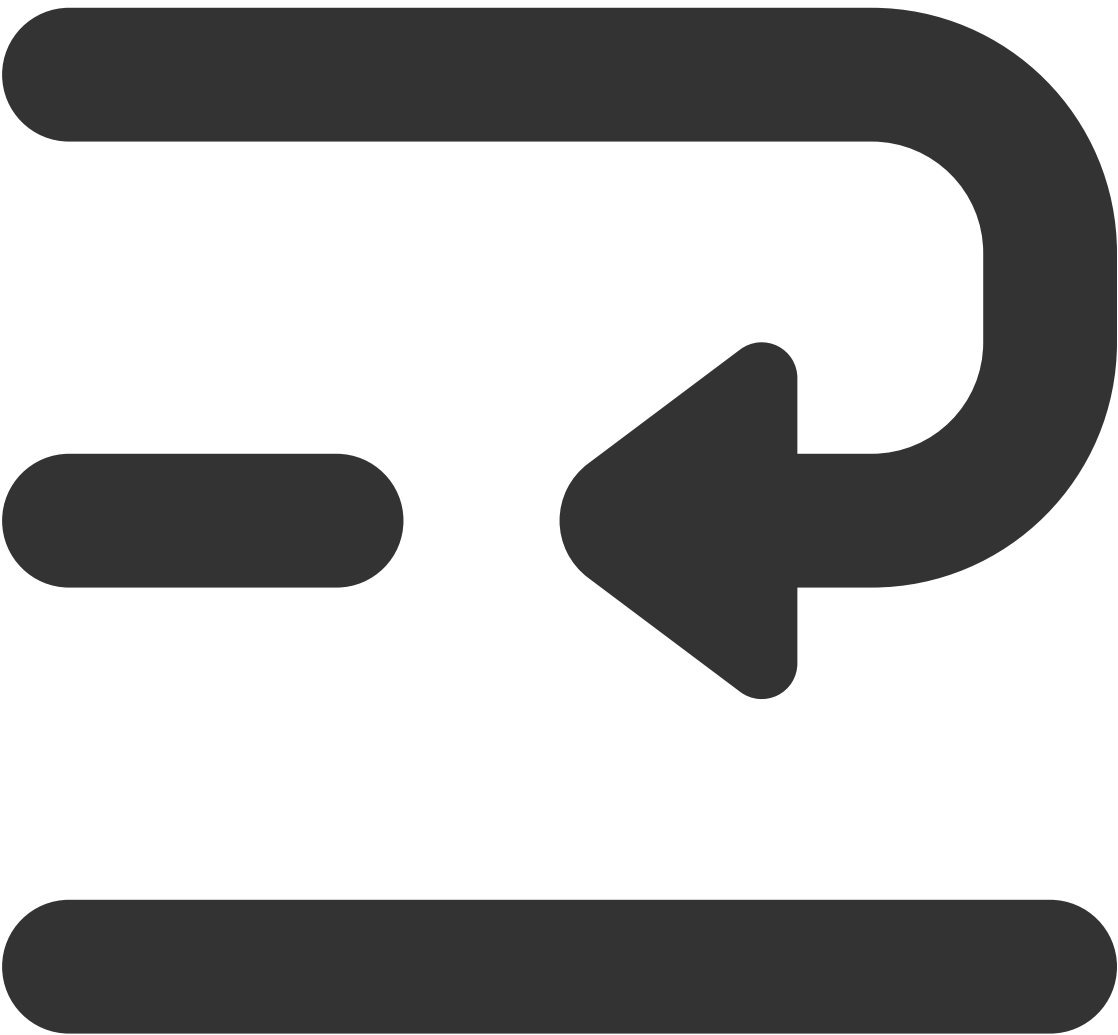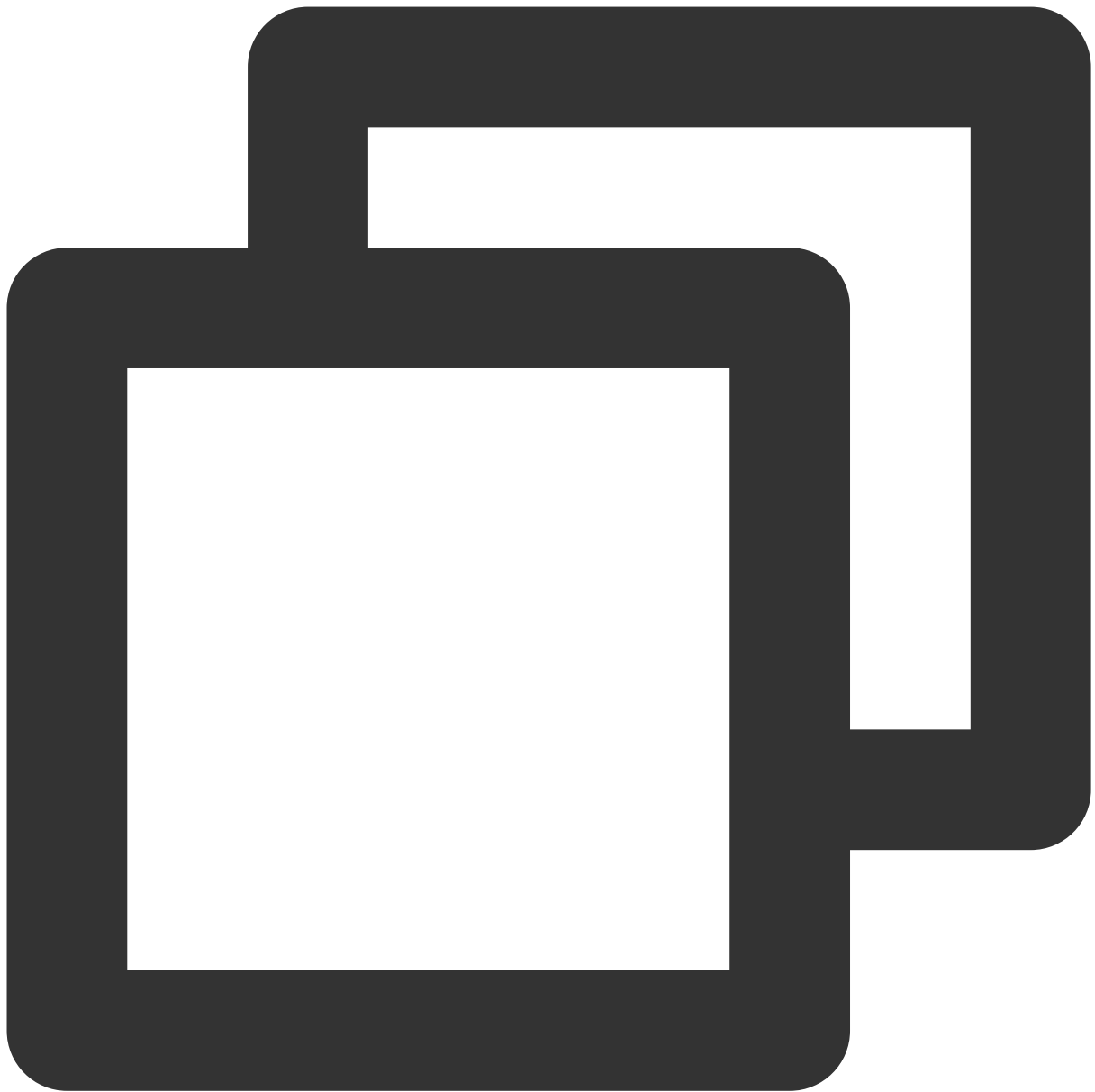
```
mkdir -p ./TUIKit/tui-customer-service-plugin && rsync -av ./node_modules/@tencentc
```

Download the TUIKit component using the npm method:
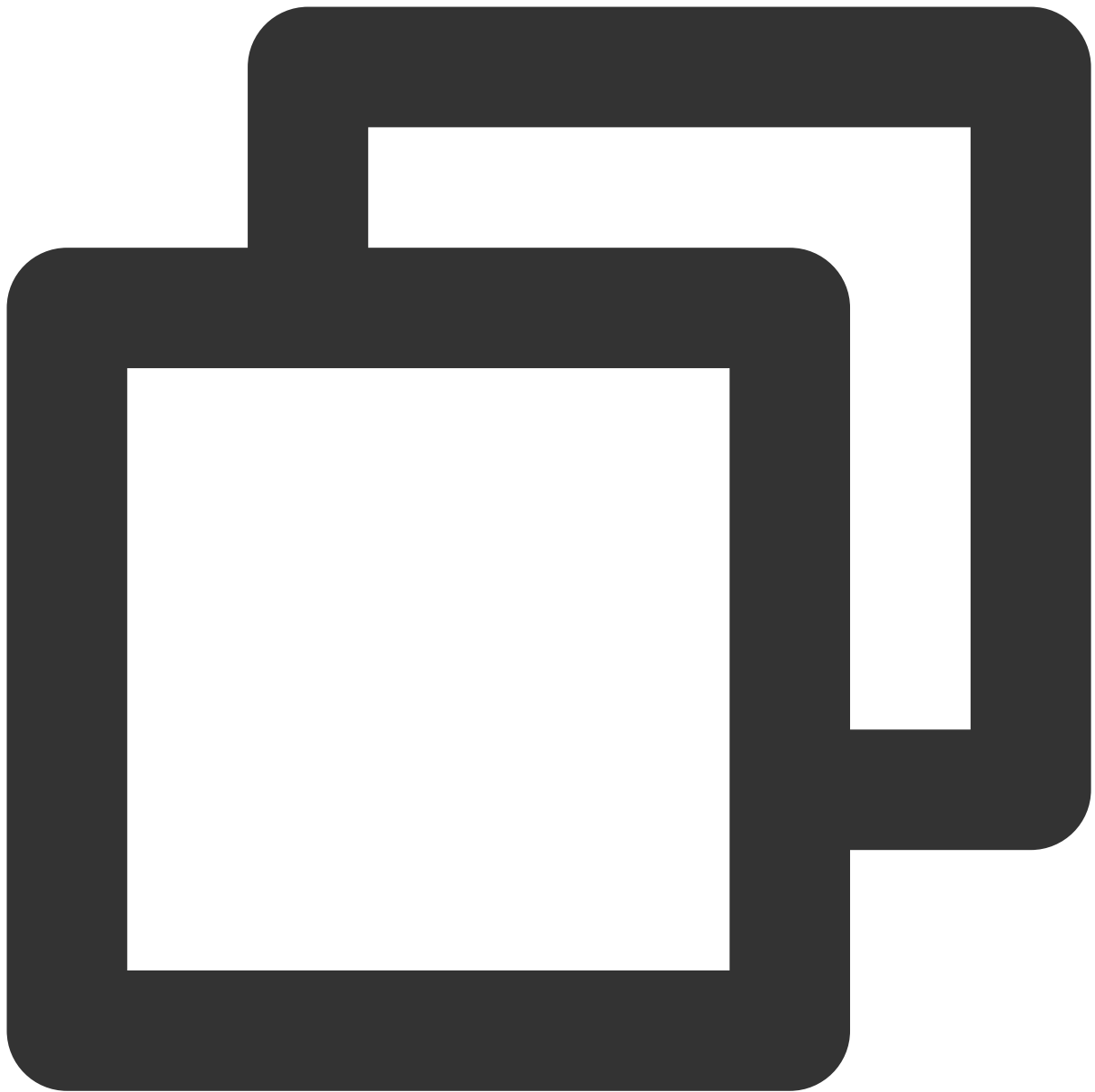
```
npm i @tencentcloud/chat-uikit-uniapp unplugin-vue2-script-setup
```

For ease of subsequent extensions, we propose that you replicate the TUIKit component to the pages directory within your project. Please conduct the following command in the root directory of your own project:
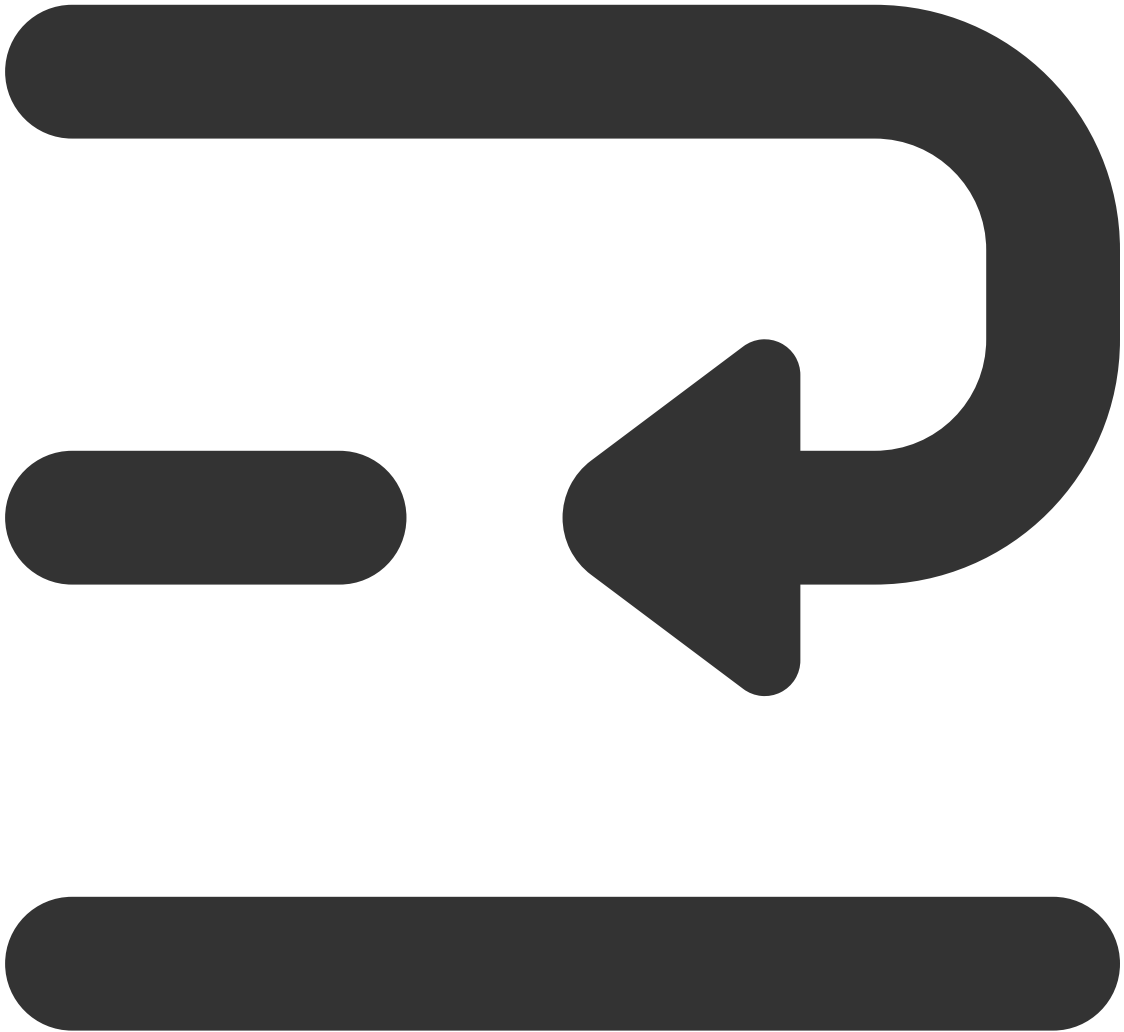
```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-uniapp .\\TUIKit /i /e /exclude:.\
```
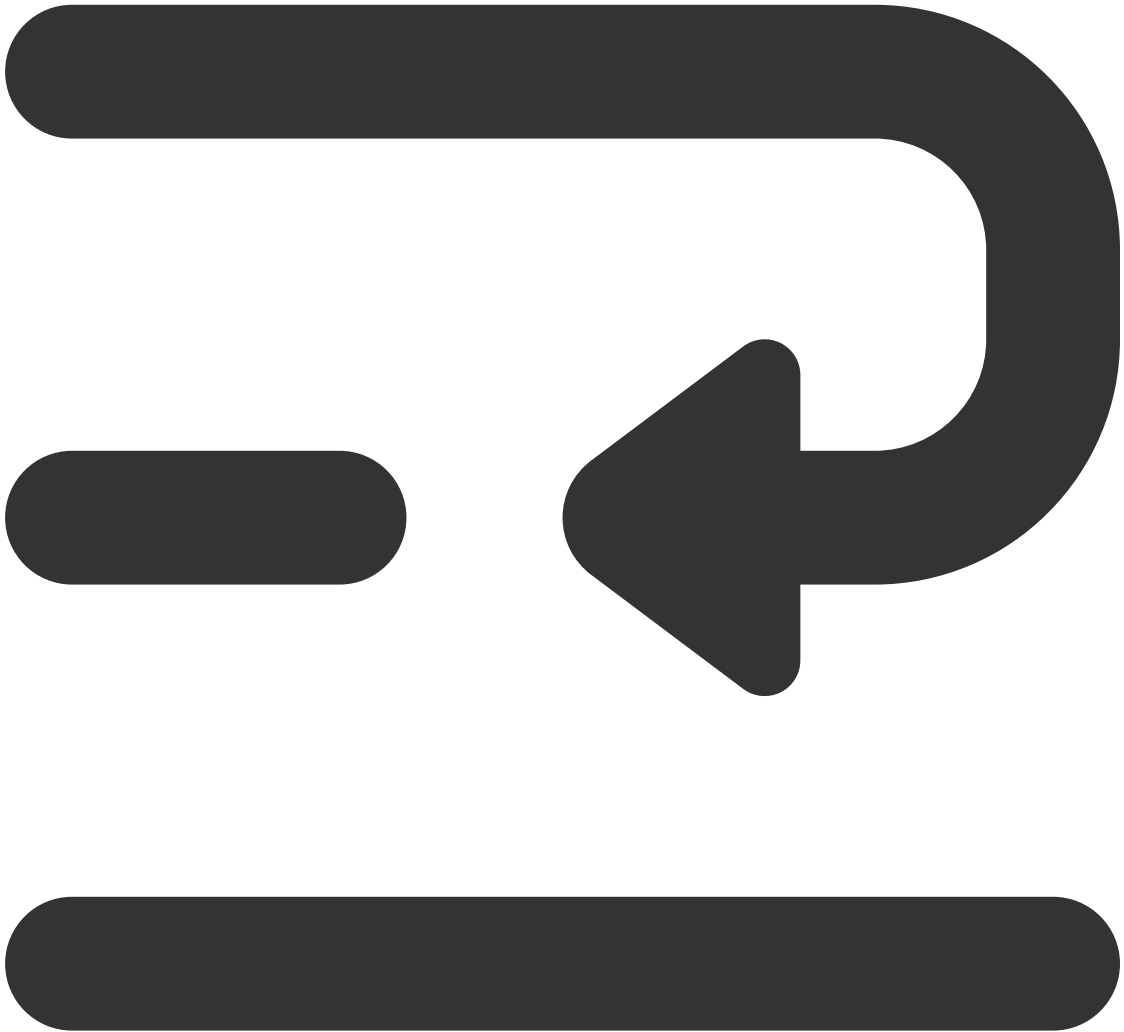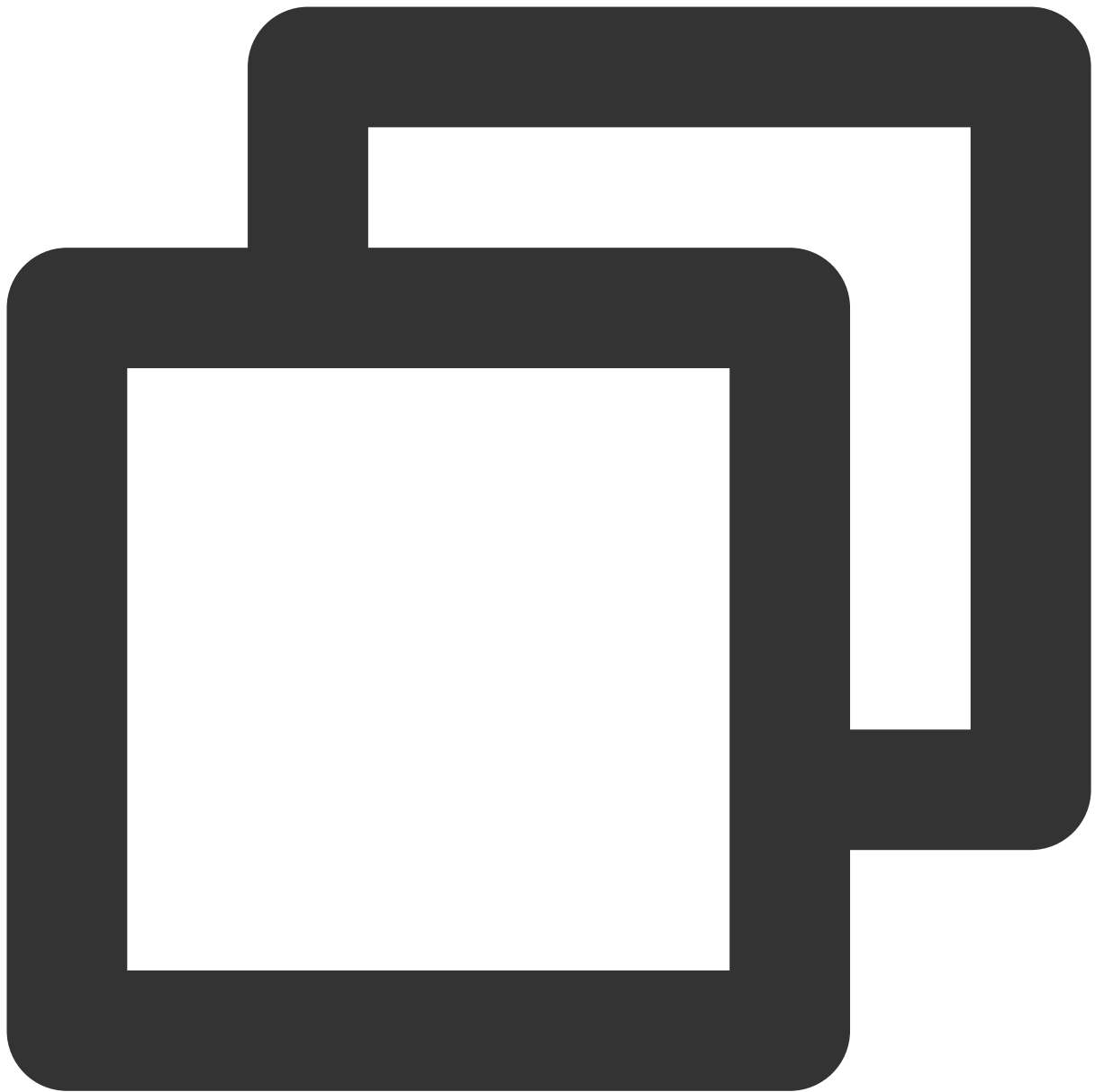
Tencent Cloud


```
xcopy .\\node_modules\\@tencentcloud\\tui-customer-service-plugin .\\TUIKit\\tui-cu
```

Step 3: Incorporate the TUIKit component

1. Project Configuration

In the root directory, create vue.config.js (For Vue3 projects, please disregard this part).
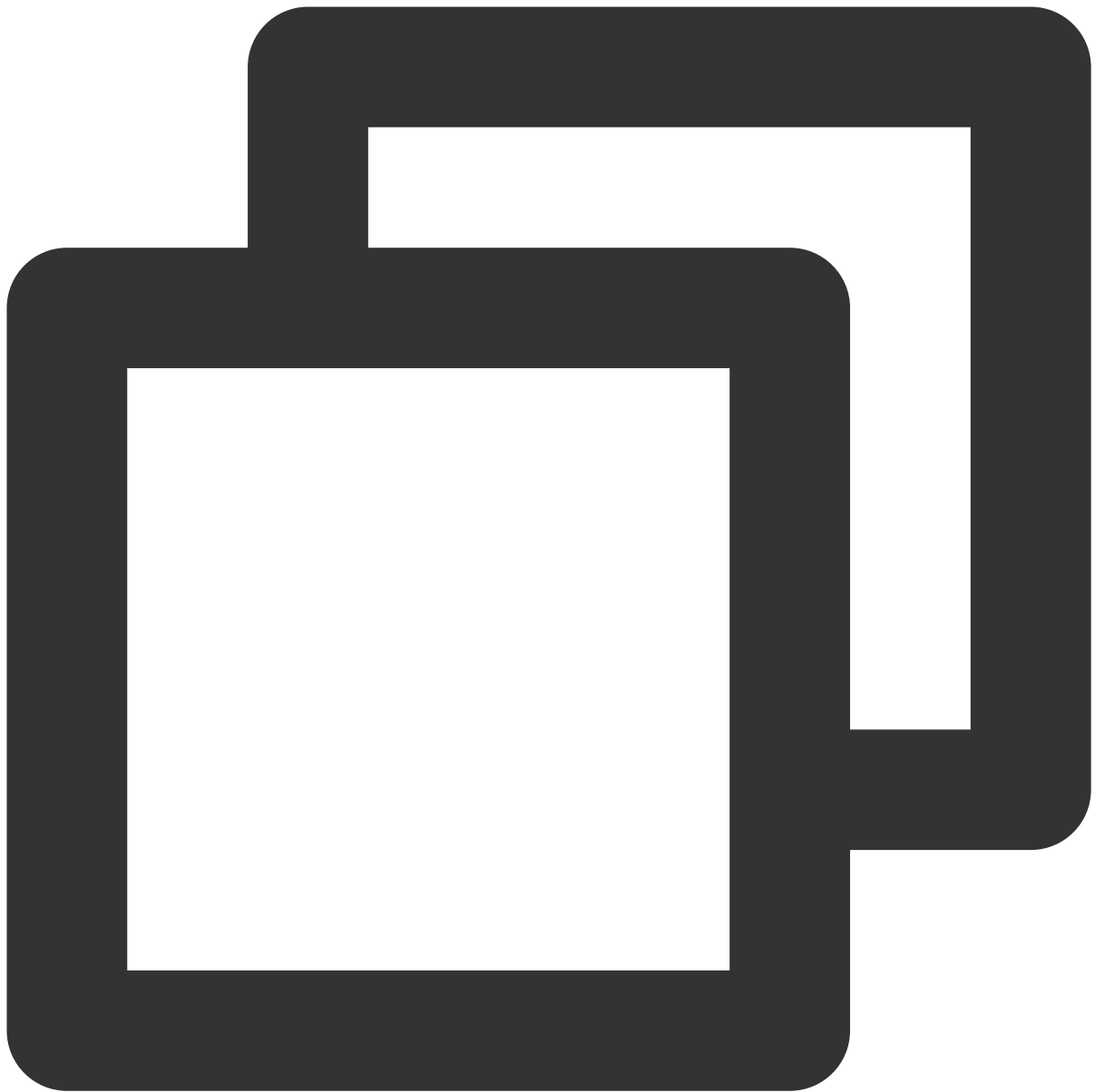
```
const ScriptSetup = require('unplugin-vue2-script-setup/webpack').default;
module.exports = {
  parallel: false,
  configureWebpack: {
    plugins: [
      ScriptSetup({
        /* options */
      }),
    ],
  },
  chainWebpack(config) {
```

```
        // disable type check and let `vue-tsc` handles it
        config.plugins.delete('fork-ts-checker');
    },
};
```

Activate the split package configuration in the source code view of the `manifest.json` file
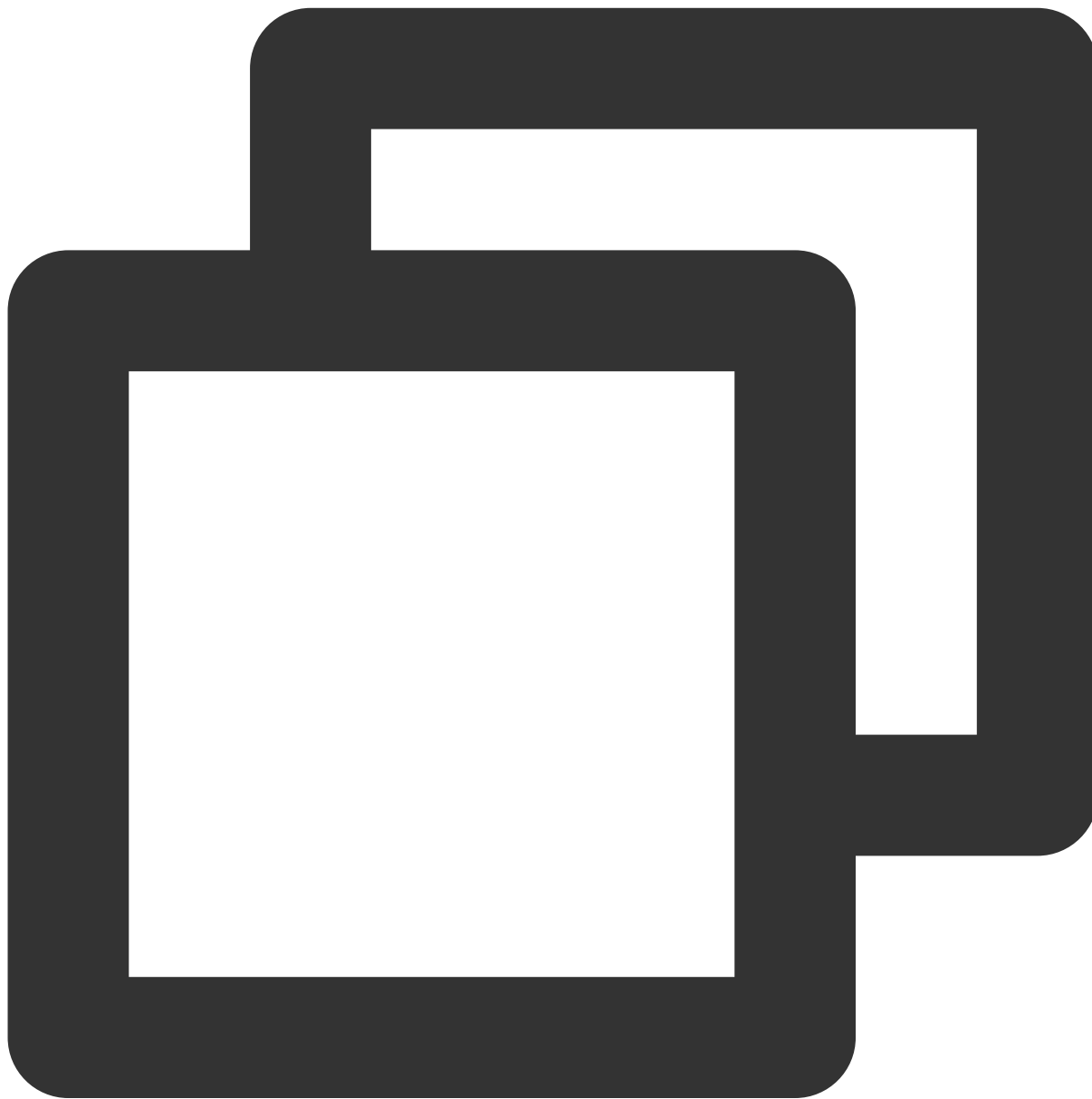

```
{
    "mp-weixin": {
        "appid": "",
        "optimization": {
```

```
            "subPackages": true
        }
    },
    "h5": {
        "optimization": {
            "treeShaking": {
                "enable": false
            }
        }
    }
}
```

2. Merge TUIKIt

Note:

Pursue the integration stringently in **Four Steps**. If you wish to package a Mini Program, please do not bypass the configuration of the "Home page of Mini Program Sub-package".

main.js file

pages.json file

App.vue file

Mini Program Sub-package Home Page

Pay heed, under Vue2 environment, make use of `Vue.use(VueCompositionAPI)` , to prevent inability to use environment variables such as `isPC` .

```
// Introduce the main package dependency
import TencentCloudChat  from "@tencentcloud/chat";
import TUICore from "@tencentcloud/tui-core";

import App from './App';

// #ifndef VUE3
import Vue from 'vue';
import './uni.promisify.adaptor';
import VueCompositionAPI from "@vue/composition-api";
Vue.use(VueCompositionAPI);
```

```
Vue.config.productionTip = false;
App.mpType = 'app';
const app = new Vue({
  ...App,
});
app.$mount();
// #endif

// #ifdef VUE3
import { createSSRApp } from 'vue';
export function createApp() {
  const app = createSSRApp(App);
  return {
    app,
  };
}
// #endif
```

```
{
 "pages": [{
  "path": "pages/index/index" // Your project's homepage
 }],
 "subPackages": [{
  "root": "TUIKit",
  "pages": [
   {
    "path": "components/TUIConversation/index",
    "style": {
     "navigationBarTitleText": "Tencent Cloud IM"
```

```
      }
    },
    {
      "path": "components/TUIChat/index",
      "style": {
        "navigationBarTitleText": "Tencent Cloud IM"
      }
    },
    // To integrate the chat component, this path must be configured: video playbac
    {
      "path": "components/TUIChat/video-play",
      "style": {
        "navigationBarTitleText": "Tencent Cloud IM"
      }
    },
    {
      "path": "components/TUIChat/web-view",
      "style": {
        "navigationBarTitleText": "Tencent Cloud IM"
      }
    },
    {
      "path": "components/TUIContact/index",
      "style": {
        "navigationBarTitleText": "Tencent Cloud IM"
      }
    },

    {
      "path": "components/TUIGroup/index",
      "style": {
        "navigationBarTitleText": "Tencent Cloud IM"
      }
    }
  ]
}],
"preloadRule": {
  "TUIKit/components/TUIConversation/index": {
    "network": "all",
    "packages": ["TUIKit"]
  }
},
"globalStyle": {
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "uni-app",
  "navigationBarBackgroundColor": "#F8F8F8",
  "backgroundColor": "#F8F8F8"
```

```
    }
  }
```
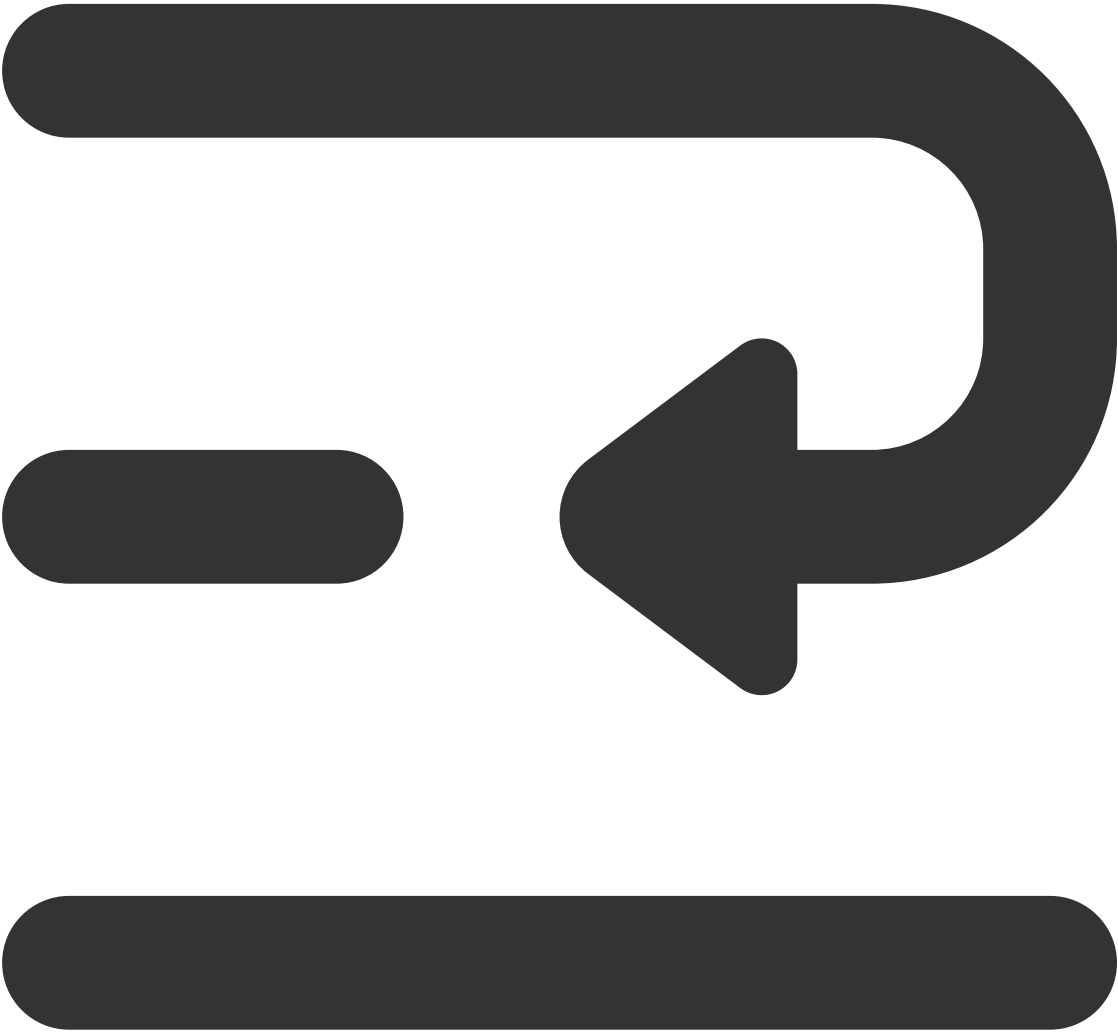
```ts
<script lang="ts">
// #ifdef APP-PLUS || H5
import { TUIChatKit, genTestUserSig } from "./TUIKit";
import { vueVersion } from "./TUIKit/adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";
// #endif
// Mandatory information
const config = {
  userID: "test-user1", // User ID
  SDKAppID: 0, // Your SDKAppID
  secretKey: "", // Your secretKey
```

```
};
uni.$chat_userID = config.userID;
uni.$chat_SDKAppID = config.SDKAppID;
uni.$chat_secretKey = config.secretKey;

// #ifdef APP-PLUS || H5
uni.$chat_userSig = genTestUserSig(config).userSig;
// Initialization of TUIChatKit
TUIChatKit.init();
// #endif
export default {
  onLaunch: function () {
    // #ifdef APP-PLUS || H5
    // TUICore login
    TUILogin.login({
      SDKAppID: uni.$chat_SDKAppID,
      userID: uni.$chat_userID,
      // A UserSig is a cipher for users to sign in to Instant Messaging - it is es
      // This method is only suitable for running demos locally and debugging featu
      userSig: uni.$chat_userSig,
      // Should you require to transmit imagery, audio, video, files, and other for
      useUploadPlugin: true,
      // Local audit can identify and handle unsuitable and unsafe content to effec
      // This feature is an added service, please refer to: https://cloud.tencent.c
      // If you've purchased the content review service, please enable this feature
      useProfanityFilterPlugin: false,
      framework: `vue${vueVersion}` // Current development framework in use: vue2 /
    });
    // #endif
  },
 onShow: function() {
     console.log('App Show')
  },
 onHide: function() {
     console.log('App Hide')
  }
};
</script>
<style>
/*Common CSS for each page*/
uni-page-body,
html,
body,
page {
  width: 100% !important;
  height: 100% !important;
  overflow: hidden;
```

```
}
</style>
```

Note:

The mini program integrates by default in a subpackage. The login must be completed on the TUIKit startup page.

If you do not require the packaging of mini-programs (for instance, building H5 only), you can disregard the configuration content of "*Mini Program Split Package Homepage*".

Example: The TUIKit sub-package first screen launch page is the **TUIConversation** page

Step 1: Create a subPackage-init.ts file under the TUIKit/components/TUIConversation directory
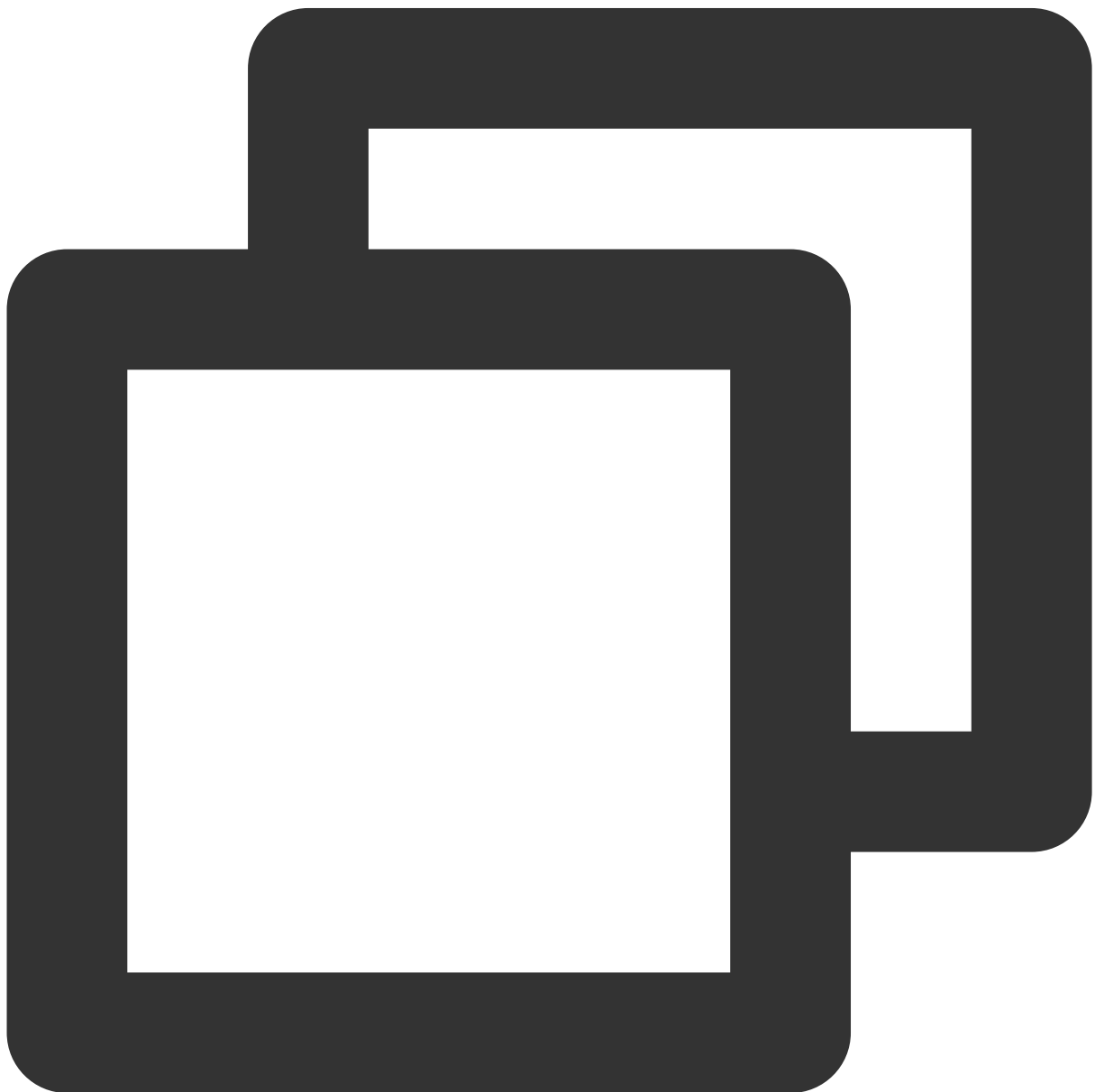
```
import { TUIChatKit, genTestUserSig } from "../../index.ts";
import { vueVersion, onMounted } from "../../adapter-vue";
import { TUILogin } from "@tencentcloud/tui-core";


// Initialization of TUIChatKit
TUIChatKit.init();
uni.$chat_userSig = genTestUserSig({
        userID: uni.$chat_userID,
        SDKAppID: uni.$chat_SDKAppID,
        secretKey: uni.$chat_secretKey
}).userSig;


// login
TUILogin.login({
  SDKAppID: uni.$chat_SDKAppID,
  userID: uni.$chat_userID,
  // UserSig is the cipher for users to sign in to Instant Messaging, essentially b
  // This method is only suitable for running Demo locally and debugging functions.
  userSig: uni.$chat_userSig,
  // Should you require to send image, voice, video, file and other rich media mess
  useUploadPlugin: true,
  // Local review can successfully identify and handle inappropriate and unsafe con
  // This functionality is a value-added service, please refer to: https://cloud.te
  // If you have purchased the content review service, to activate this feature ple
  useProfanityFilterPlugin: false,
  framework: `vue${vueVersion}` // Current development uses framework vue2 / vue3
}).then(() => {
  uni.showToast({
    title: "login success"
  });
});
```

Step 2: Import within TUIKit/components/TUIConversation/index.vue

Tencent Cloud


```
// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

See the following figure:

```
<script lang="ts" setup>
import {
  TUIStore,
  TUIGlobal,
  StoreName,
} from "@tencentcloud/chat-uikit-engine";
import { ref } from "../../adapter-vue";
import ConversationList from "./conversation-list/index.vue";
import ConversationHeader from "./conversation-header/index.vue";
import ConversationNetwork from "./conversation-network/index.vue";
import { onHide } from "@dcloudio/uni-app"; // 该方法只能用在父组件内，子组件内不生效

// #ifdef MP-WEIXIN
import "./subPackage-init.ts";
// #endif
```

3. Configuring the entry points of TUIConversation and TUIContact on the main package homepage of the project

Create an index.vue file under the pages/index folder

```
<template>
  <div class="index">
        <p class="index-button" @click="openConversation">Open TUIKit Conversation<
        <p class="index-button" @click="openContact">Open TUIKit Contacts</p>
  </div>
</template>
<script>
export default {
methods: {
  // Open the TUIKit session list
  openConversation() {
```

```
      uni.navigateTo({
          url: "/TUIKit/components/TUIConversation/index",
        });
  },
  // Accessing TUIKit Contacts
  openContact() {
    uni.navigateTo({
        url: "/TUIKit/components/TUIContact/index",
      });
  },
 },
};
</script>
<style lang="scss" scoped>
.index {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  &-button {
    width: 180px;
      padding: 10px 40px;
      color: #fff;
      background-color: #006eff;
      font-size: 16px;
      margin-top: 65px;
      border-radius: 30px;
      text-align: center;
  }
 }
</style>
```

Step 4: Gain access to SDKAppID, secretKey, and userID

Within the App.vue file in the root directory of configuration, find `config` object's SDKAppID, secretKey, and userID. The SDKAppID and secretKey can be accessed through the Instant Messaging Console, and the userID can be accessed when creating an account in the Instant Messaging Console.
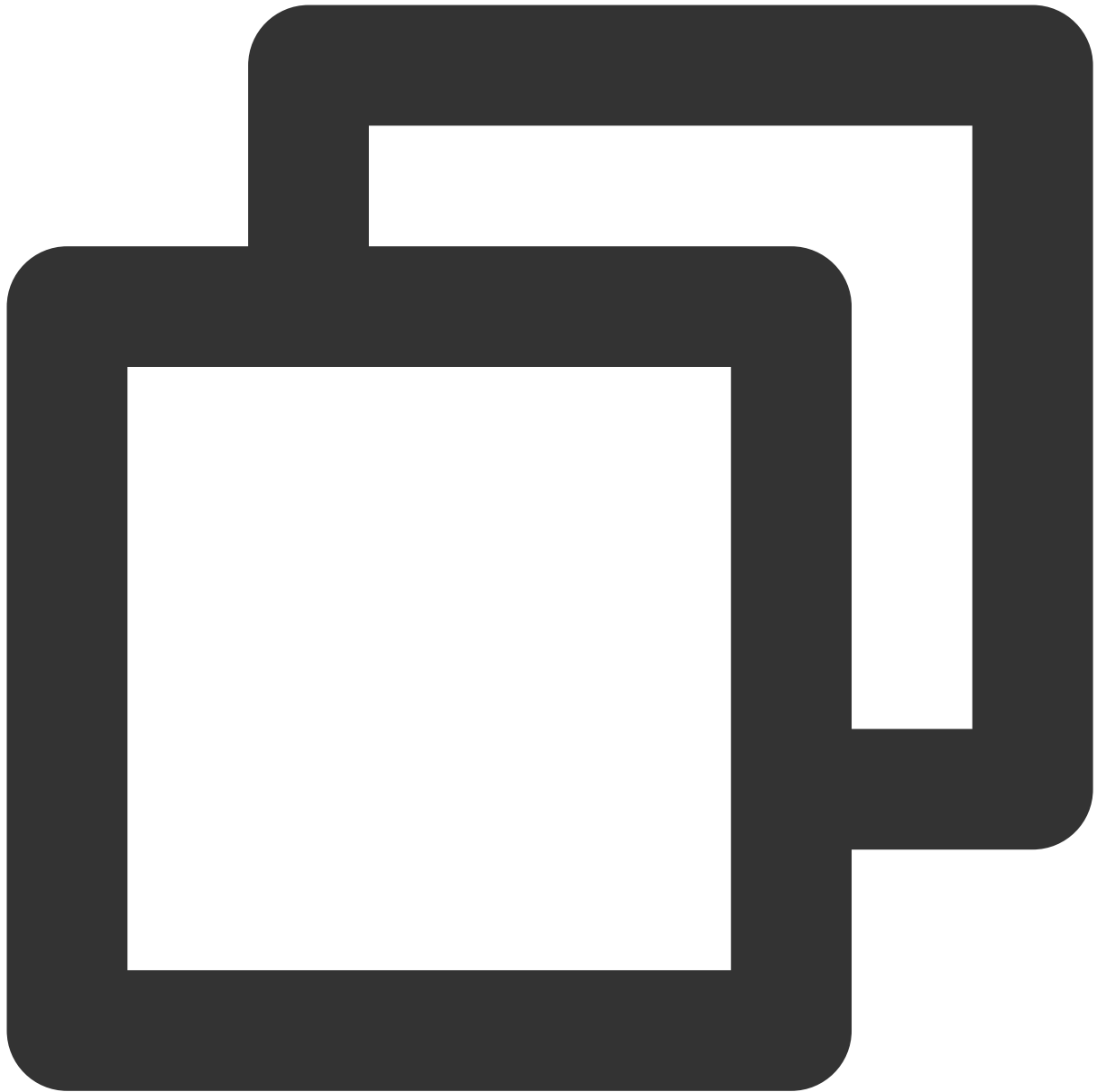
```
// Mandatory information
const config = {
  userID: "test-user1", // Login User ID
  SDKAppID: 0, // Your SDKAppID
  secretKey: "", // Your secretKey
};
```

access SDKAppID,secretKey

In the Instant Messaging Console under the **application management** page, you can see the applications you have created. The SDKAppID is in the second column. Then click on the **peekKey** in the operation options. A dialogue box will appear on the website for the peekKey, and by clicking on the **Show Key**, the peekKey will be revealed.

Create an account with `userID` as `test-user1`

Click on **Account Management** on the left side of the console. If you have multiple applications, ensure to switch to your current application. Then, under the current application, click **Create new account** to create an account with a userID of `test-user1` .

Note:

The step of creating an account can be circumvented as TUIKit will auto-generate an account during the sign in process if the configuration's userID does not exist. This only demonstrates how to access the userID.

Step 5. Launch the project

1. Launch the project using HBuilderX, then click on "Run - Run to Mini Program Simulator - WeChat Developer Tools".
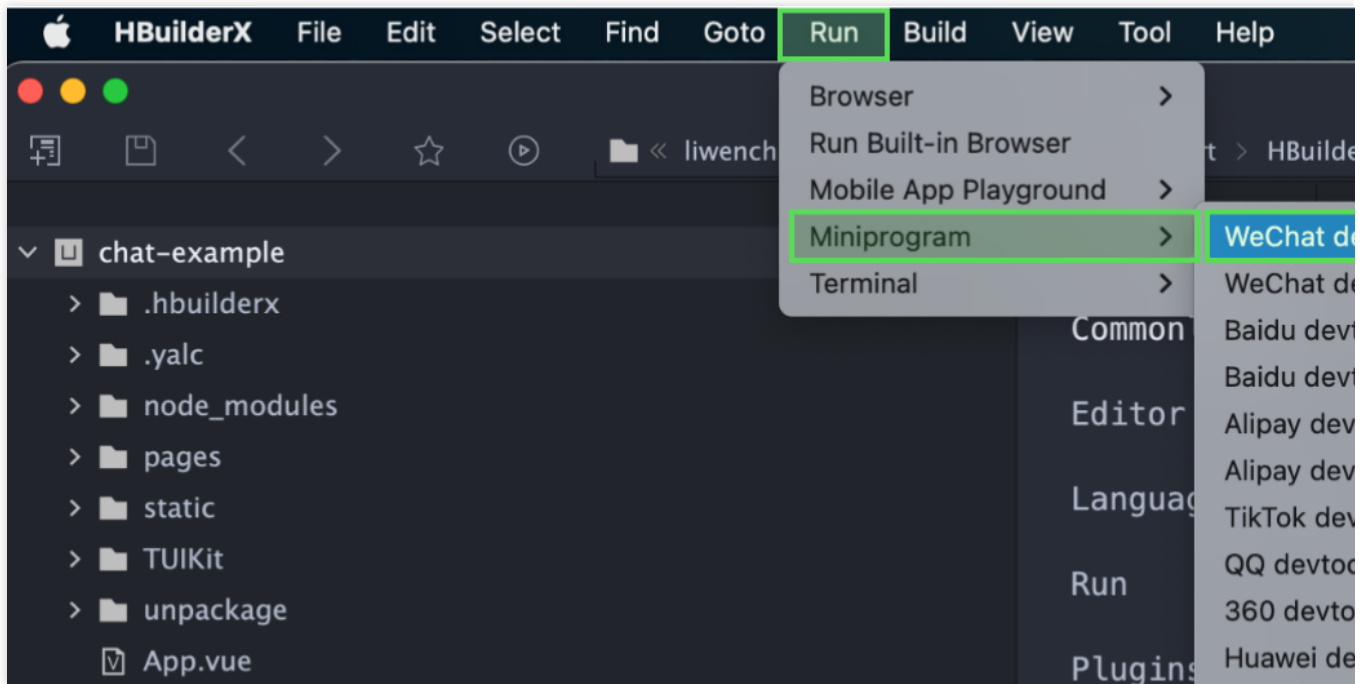
2. Should HBuilderX fail to automatically activate the WeChat Developer Toolkit, kindly use the toolkit to manually open the compiled project.

Open the `unpackage/dist/dev/mp-weixin` under the project root directory using the WeChat Developer Tool.

3. After opening the project, check the "Do not verify valid domain, web-view (business domain), TLS version, and HTTPS certificate" in "Details-Local Setting" of WeChat Developer Tools.

Step 6. Send your first message

1. Create a User account through the Instant Messaging Console

Navigate to the **Account Management** page from the left sidebar, and click on **New Account** to create a regular account with userID:test-user2.
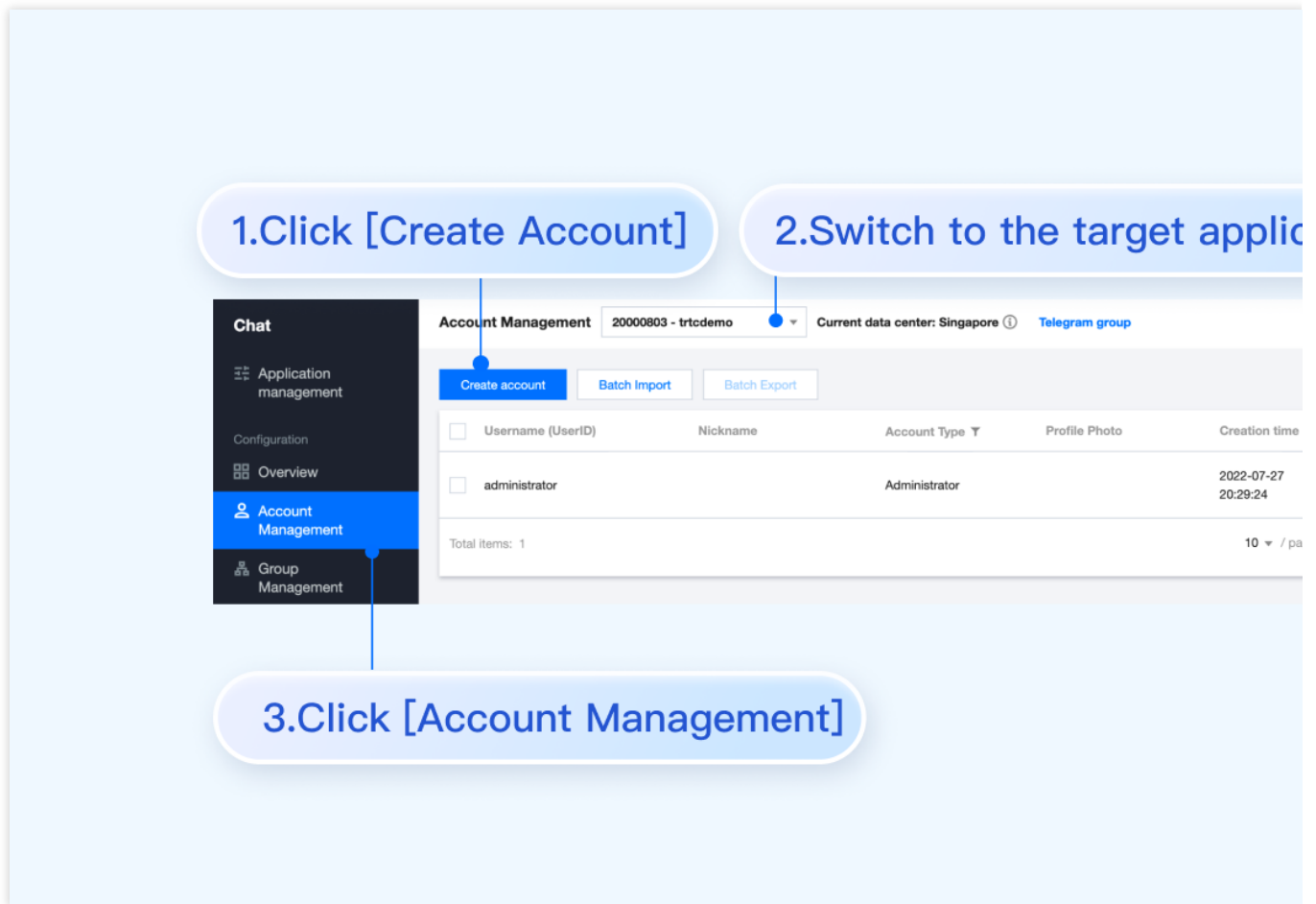
2. Run project and create conversation.

click **to open TUIKit conversation** , search for user userID:test-user2, and send your first message.
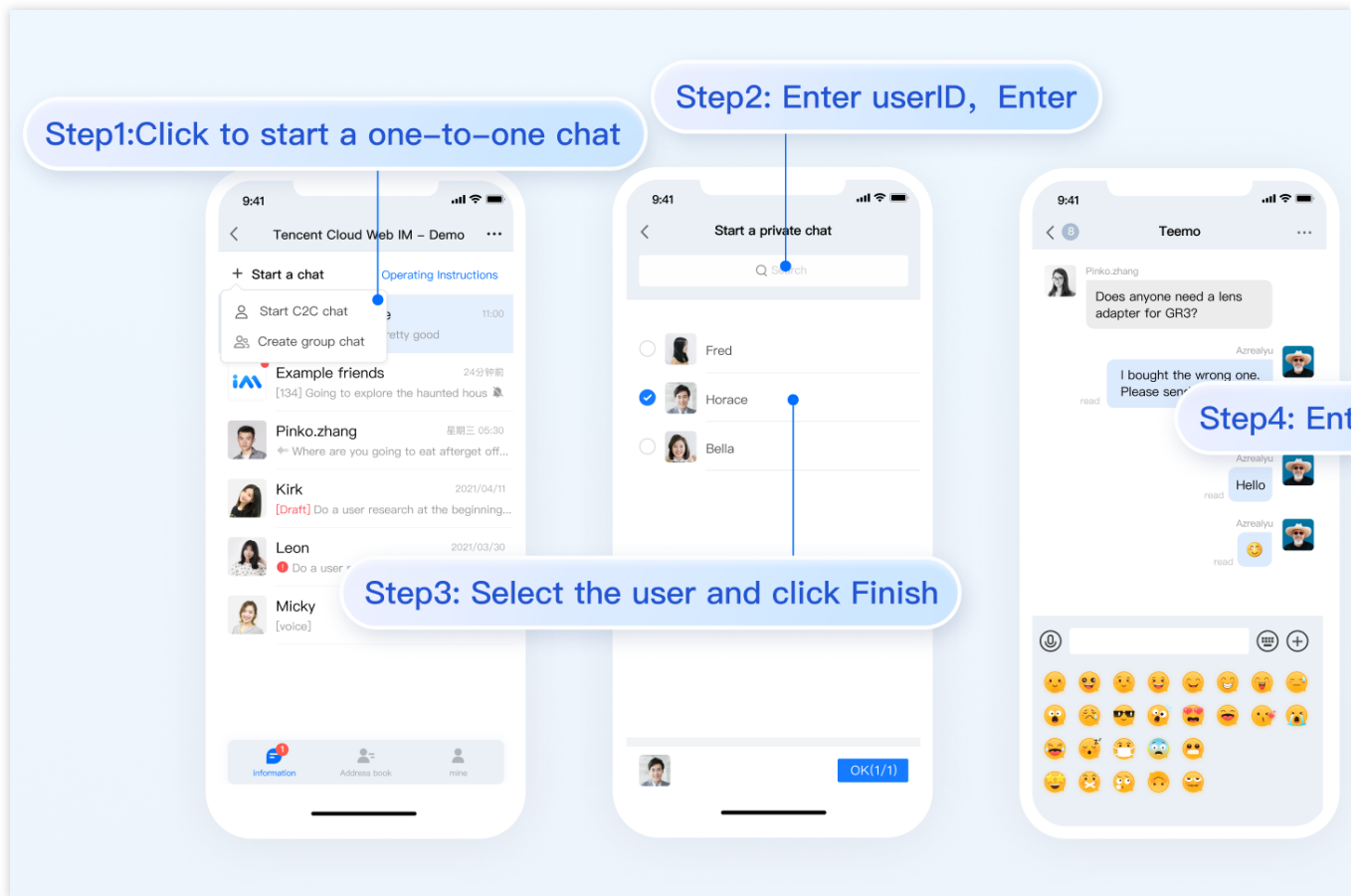
Additional Advanced Features

Audio-Visual Communication TUICallKit Plugin

Note:

The TUICallKit audio/video component is not integrated by default in TUIKit,TUICallKit primarily handles voice and video calls.

Should you need to integrate call functionalities, kindly refer to the following documents for guidelines.

For packaging into APP, refer to: Audio/Video Calling (Client)

For packaging to Miniprogram, please refer to: Video Calls(Miniprogram)

For packaging into HTML5, please refer to the official documentation: Audio and Video Calls (HTML5)

Please stay tuned.

TIMPush Offline Push Plugin

Indication

By default, TUIKit does not integrate the TIMPush **offline push plugin**.TIMPush is Tencent Cloud's Instant Messaging Push Plugin. Currently, offline push supports Android and iOS platforms, and devices include: Huawei,

Xiaomi, OPPO, Vivo, Meizu, and Apple phones.

Should you require the integration of offline push capabilities within your APP, kindly refer to the implementation of uni-app offline push.

Please stay tuned.

Individually integrate TUIChat component

Consider Independent Integration of TUIChat Component as a Solution

FAQs

For additional inquiries, please refer to Uniapp FAQ.

Exchange and Feedback

Click here to join the IM community, where you'll receive support from experienced engineers to help overcome your challenges.

Reference Documentation

Related to UIKit (vue2 / vue3):

Source code of chat-uikit-uniapp (vue2/vue3) on GitHub

Rapid Incorporation of chat-uikit-uniapp npm

Regarding ChatEngine:

ChatEngine API Manual

ChatEngine npm

Web & H5（Vue）

Last updated：2024-01-31 17:47:49

TUIKit Overview

TUIKit is a UI component library based on Tencent Cloud Chat SDK. It provides universal UI components, including conversations, chats, relationship chains, groups, and audio/video calls and other features.

With these UI components, you can quickly build your own in-app chat.

While implementing UI functions, the components in TUIKit will call the corresponding interfaces of the Chat SDK to implement Chat-related logic and data processing. Therefore, developers only need to focus on their own business or personalized expansion when using TUIKit.

TUIKit Components

TUIKit is mainly divided into several UI sub-components: TUIChat, TUIConversation, TUIGroup, TUIContact, and TUISearch.

Each UI component is responsible for displaying different content.

TUIKit for Web

1. TUIChat

2. TUIConversation

3. TUICallKit

4. TUIContact

TUIKit for H5

2.TUIChat

1.TUIConversation

3.TUICallKit

4.TUIContact

5.TUISearch
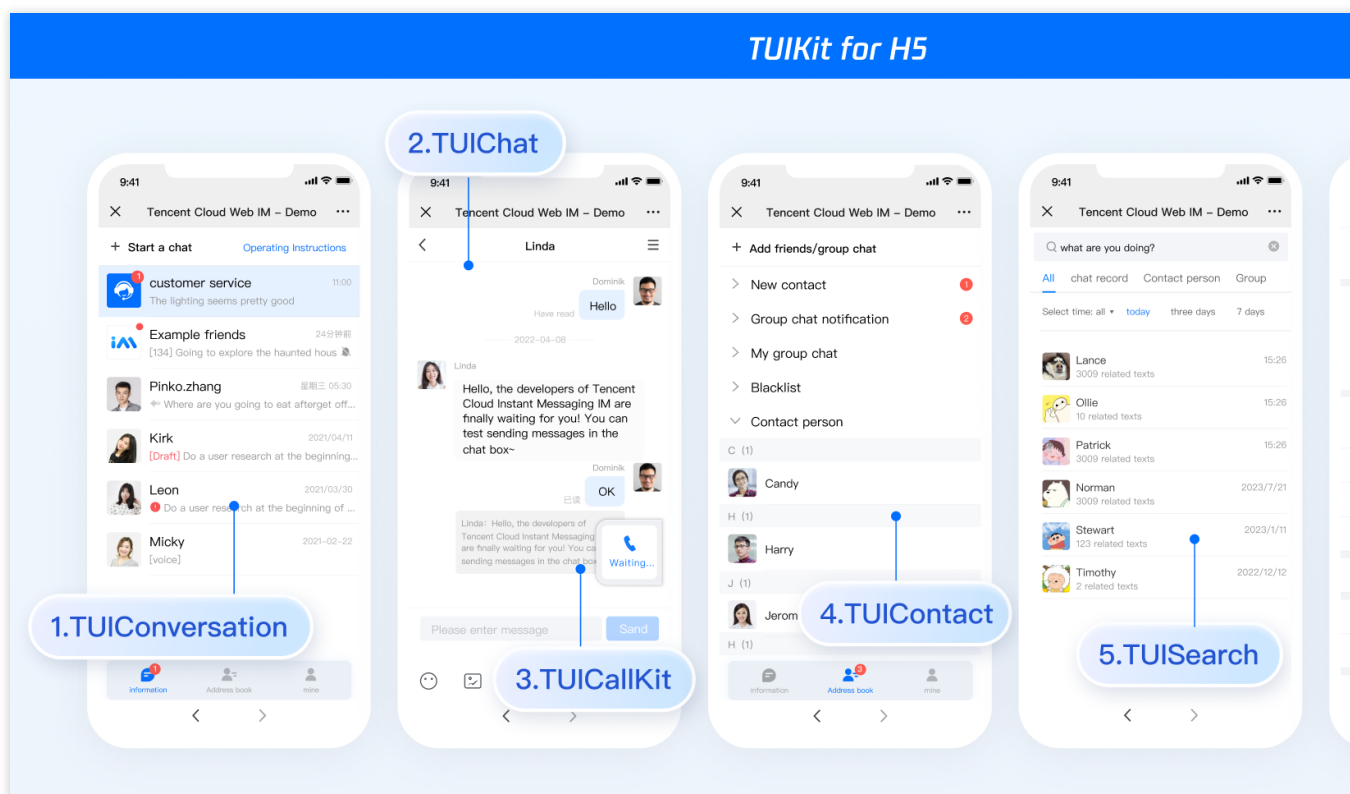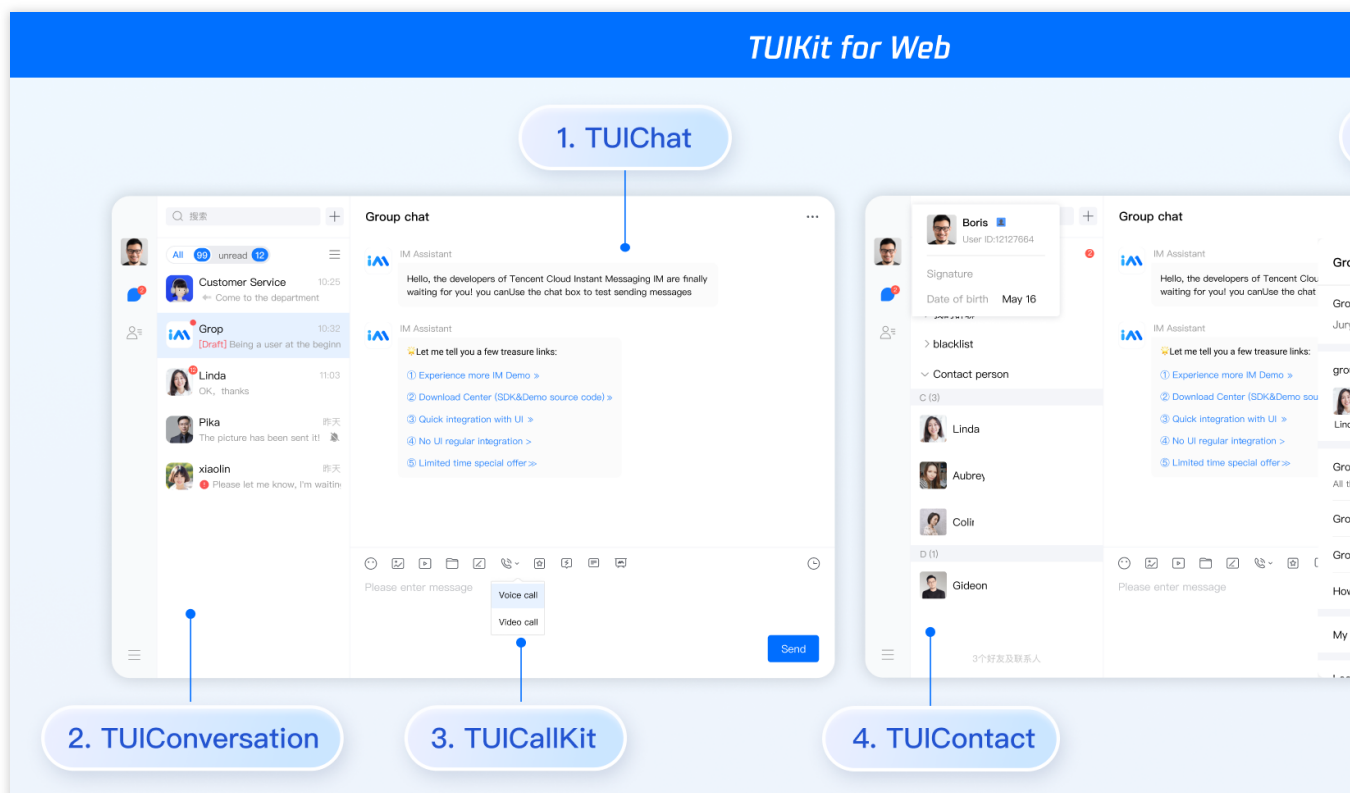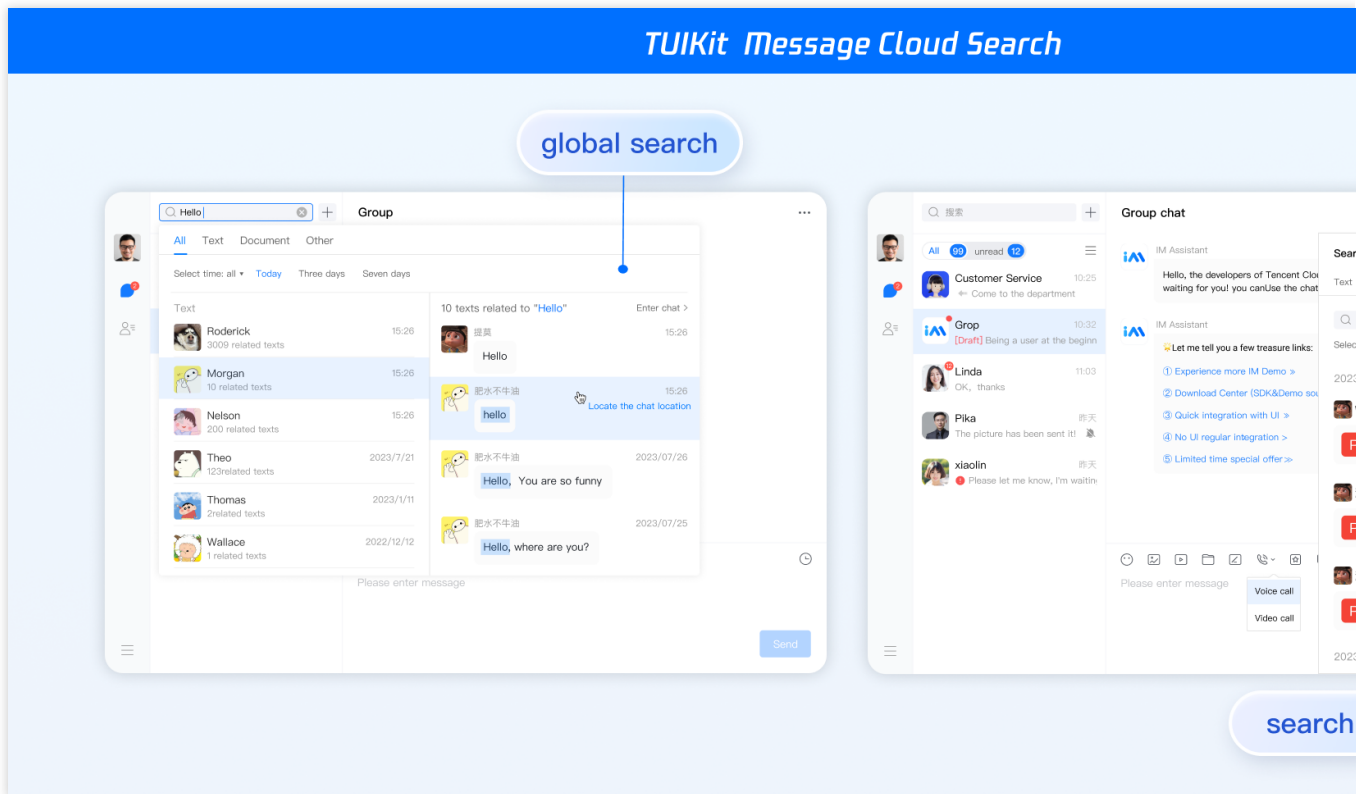
Environment Requirements

Vue (Fully compatible with both Vue2 & Vue3. While incorporating below, please select the Vue version guide that matches your needs)

TypeScript (Should your project be based on JavaScript, please proceed to JS project integrate to set up a progressive support for TypeScript)

Sass (sass-loader ≤ 10.1.1)

node(node.js ≥ 16.0.0)

npm (use a version that matches the Node version in use)

Integration of TUIKit (Web & H5)

Step 1. Create a project

TUIKit supports creating a project structure using webpack or vite, configured with Vue3 / Vue2 + TypeScript + sass.

Below are a few examples of how to construct your project:

vue-cli

vite

Please Note:

Tencent Cloud

Please make sure you have **@vue/cli version 5.0.0 or above** . The following sample code can be used to upgrade your @vue/cli version to v5.0.8.

Establish a project using Vue CLI, with configuration set to Vue2/Vue3 + TypeScript + Sass/SCSS.

If Vue CLI is not yet installed, or the version is below 5.0.0, you can use the following method for installation via Terminal or CMD:
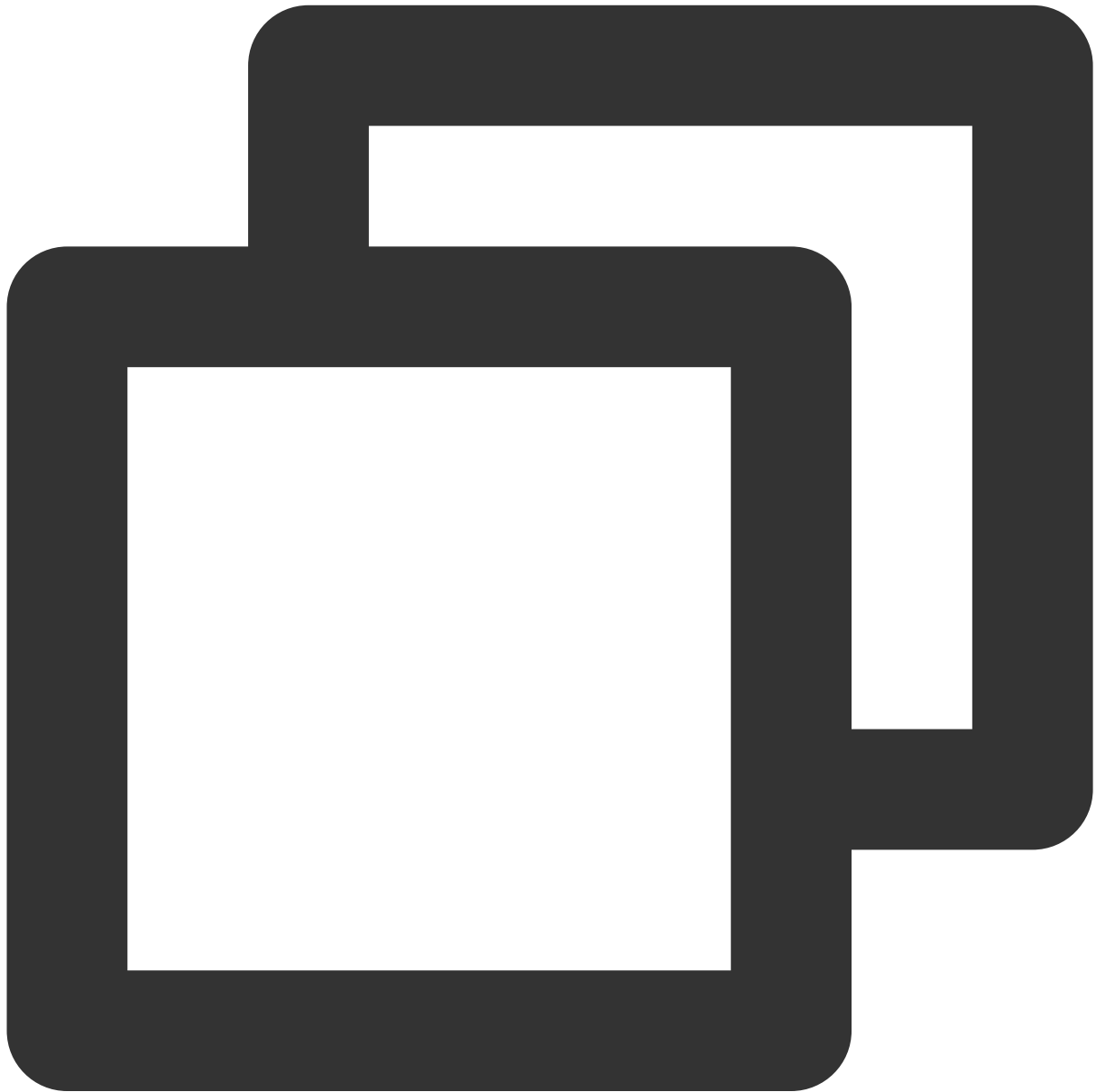
```
npm install -g @vue/cli@5.0.8 sass sass-loader@10.1.1
```

Create a project through Vue CLI and select the configuration items depicted below.

Tencent Cloud
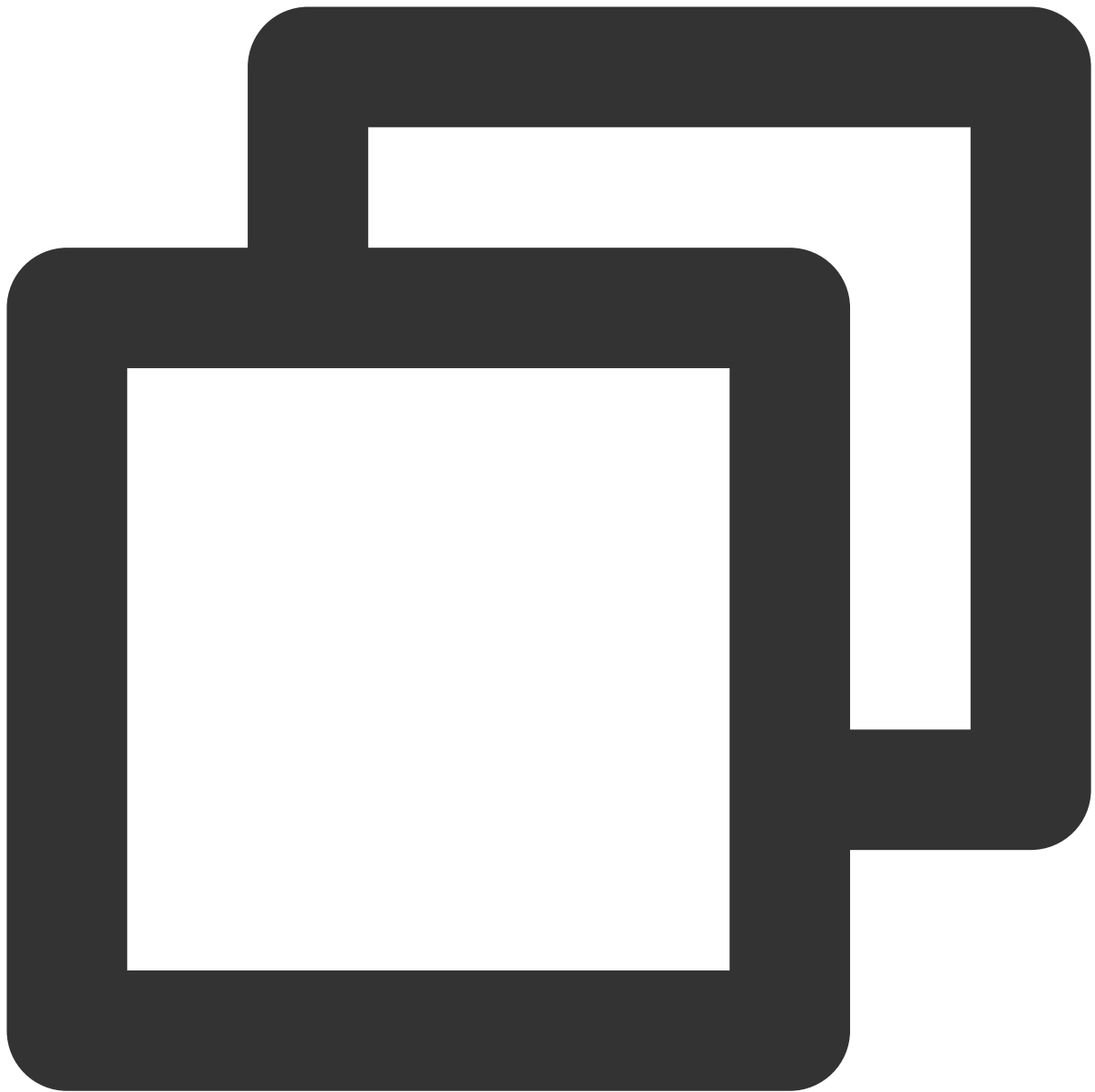

```
vue create chat-example
```

Please make sure to select according to the following configuration:

Tencent Cloud

```
Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
❯ Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to
 ◉ Babel
 ◉ TypeScript
 ◯ Progressive Web App (PWA) Support
 ◯ Router
 ◯ Vuex
❯◉ CSS Pre-processors
 ◉ Linter / Formatter
 ◯ Unit Testing
 ◯ E2E Testing
? Choose a version of Vue.js that you want to start the project with (Use arrow keys)
❯ 3.x ⏎If you need to create a vue3 project, please choose 3.x
  2.x ⏎If you need to create a vue2 project, please choose 2.x
? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): (Use
❯ Sass/SCSS (with dart-sass)
  Less
  Stylus

? Pick a linter / formatter config: (Use arrow keys)
❯ ESLint with error prevention only
  ESLint + Airbnb config
  ESLint + Standard config
  ESLint + Prettier

? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selec
❯◉ Lint on save
 ◯ Lint and fix on commit

? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
❯ In dedicated config files
  In package.json
? Save this as a preset for future projects? No
```

After creation, switch to the directory where the project is located:

Tencent Cloud
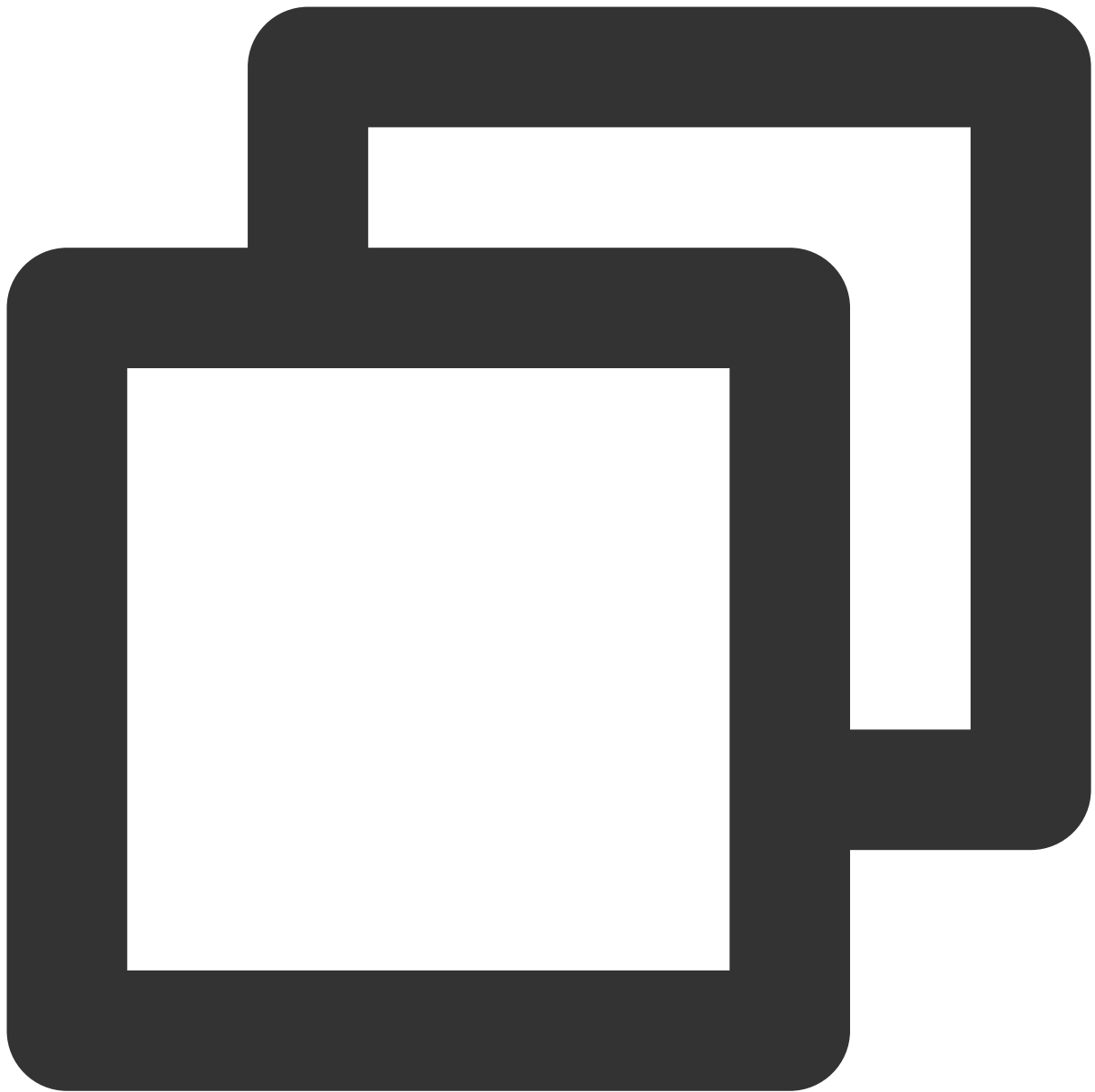
```
cd chat-example
```

If you are a vue2 project, please make the following corresponding environment configurations based on the Vue version you are using.

IIf you are a vue2 project, please ignore.

vue2.7
Vue 2.6 and below
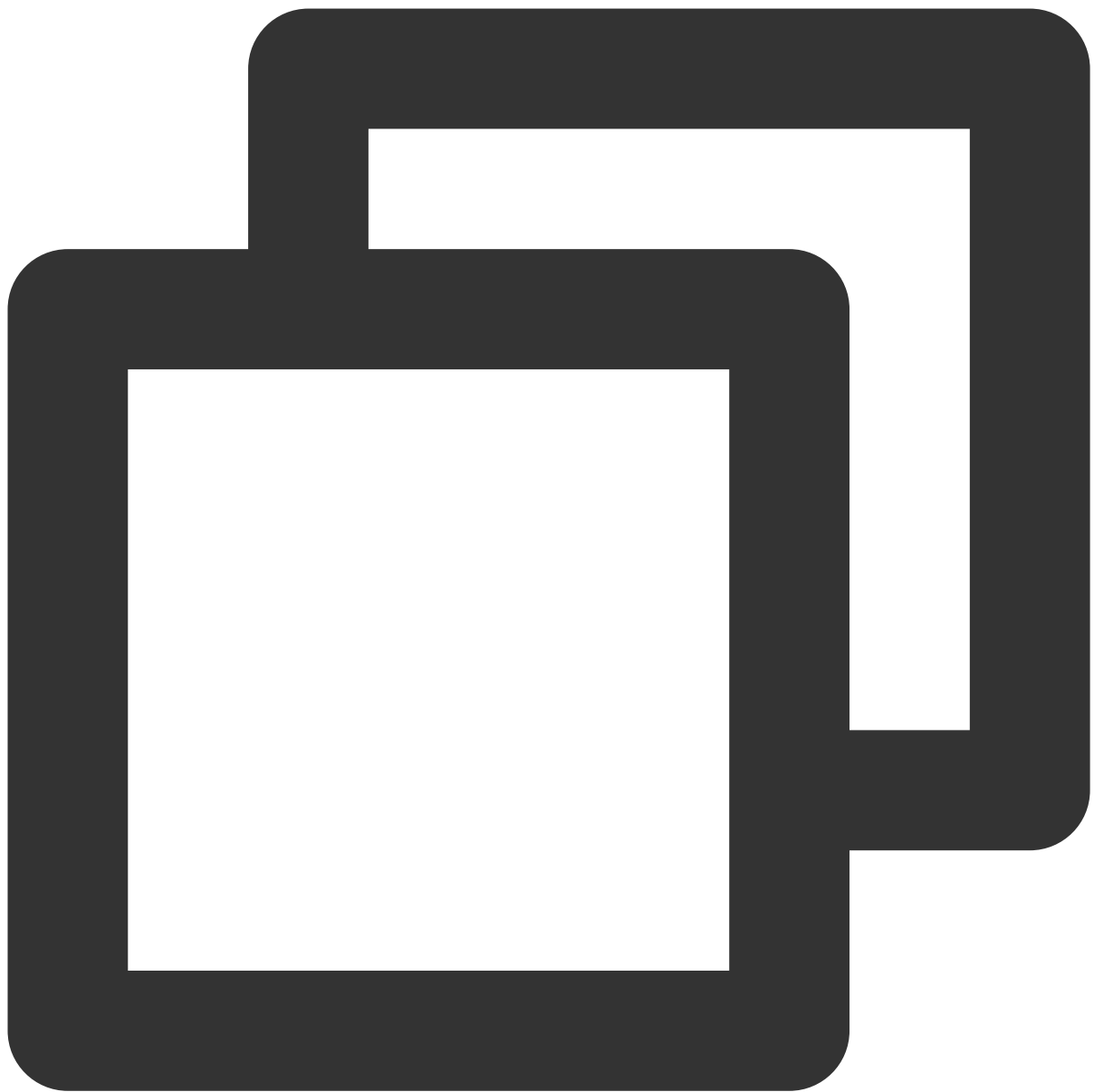
Tencent Cloud

```
npm i vue@2.7.9 vue-template-compiler@2.7.9
```

Tencent Cloud
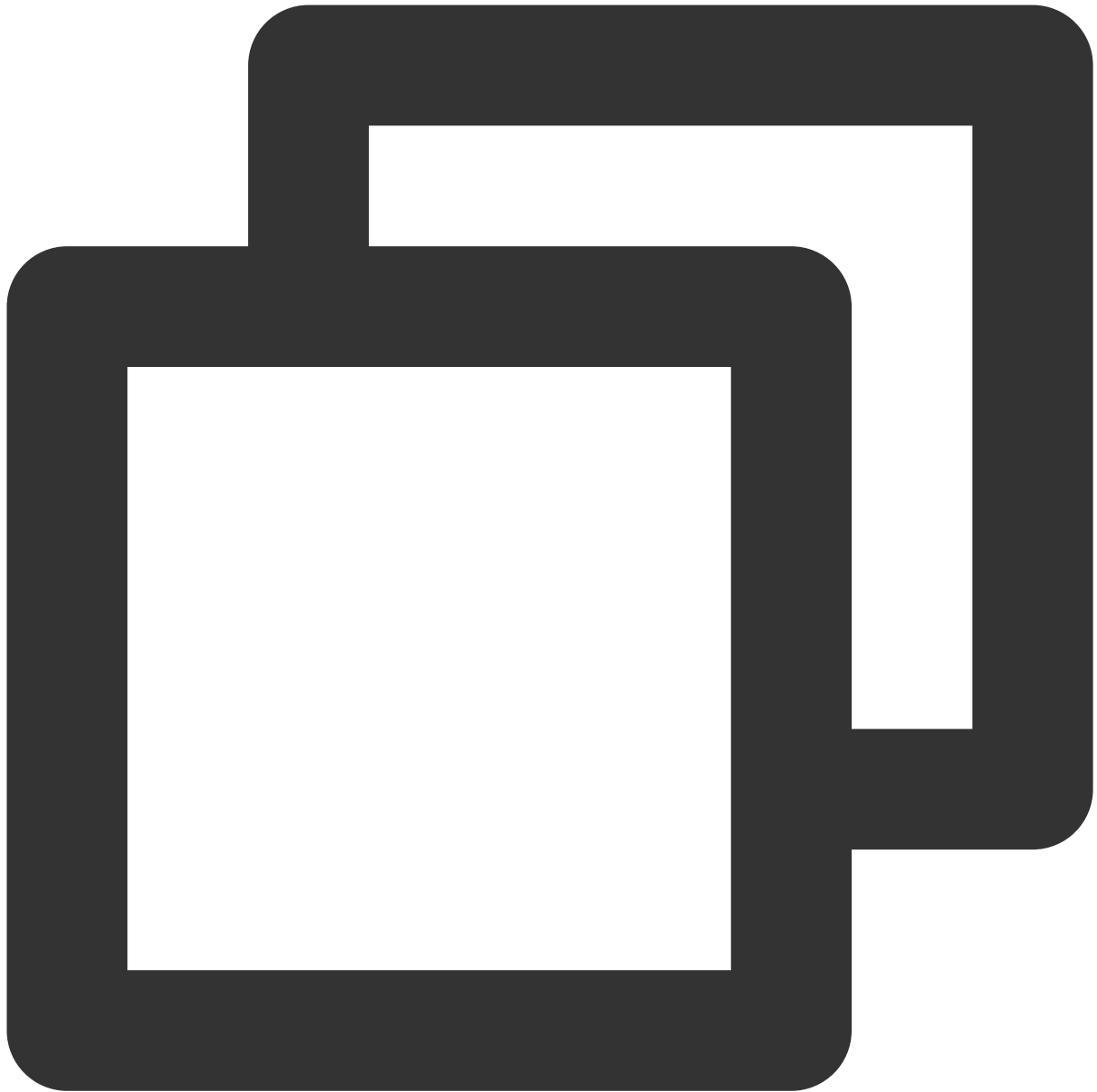
```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

Please Note:

Vite requires **Node.js versions 18+, 20+.** Pay attention to upgrade your Node version when your package manager issues a warning, for more details refer to Vite official website.

Create a project using Vite, configure Vue + TypeScript according to the options in the picture below.
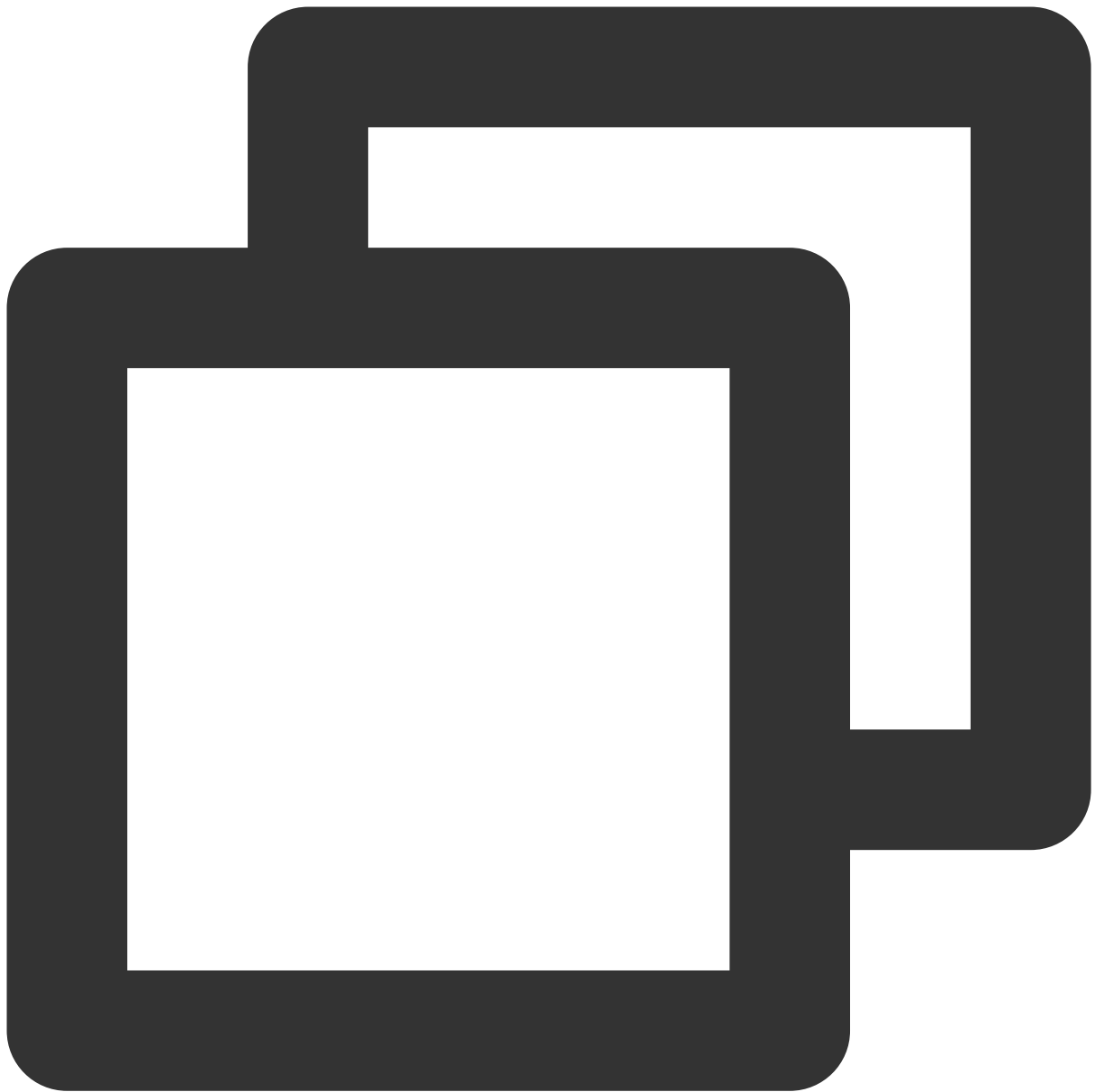
```
npm create vite@latest
```


Then, switch to the project directory, and install the project dependencies:
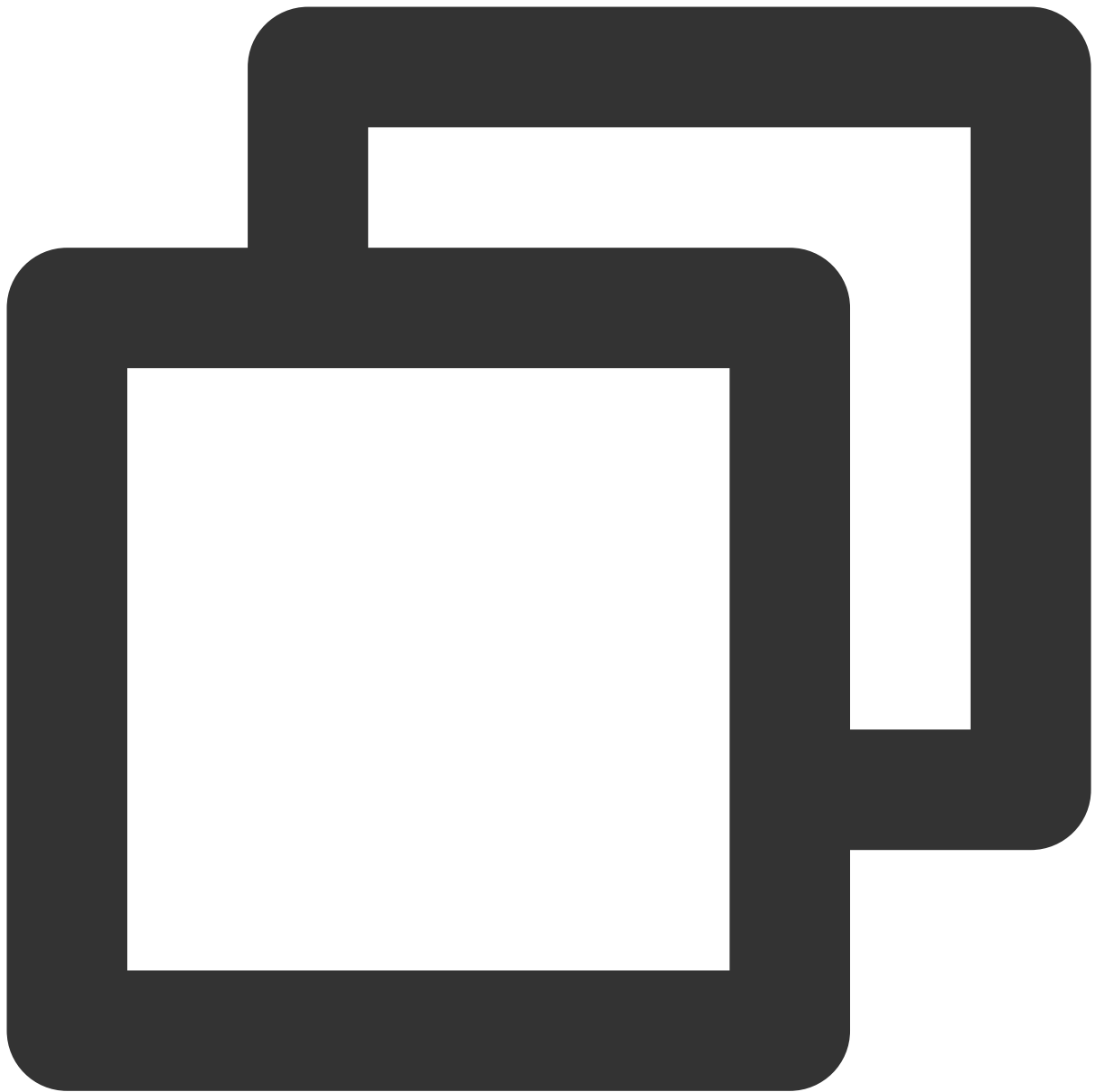
```
cd chat-example
npm install
```

Install the sass environment dependency required for TUIKit:
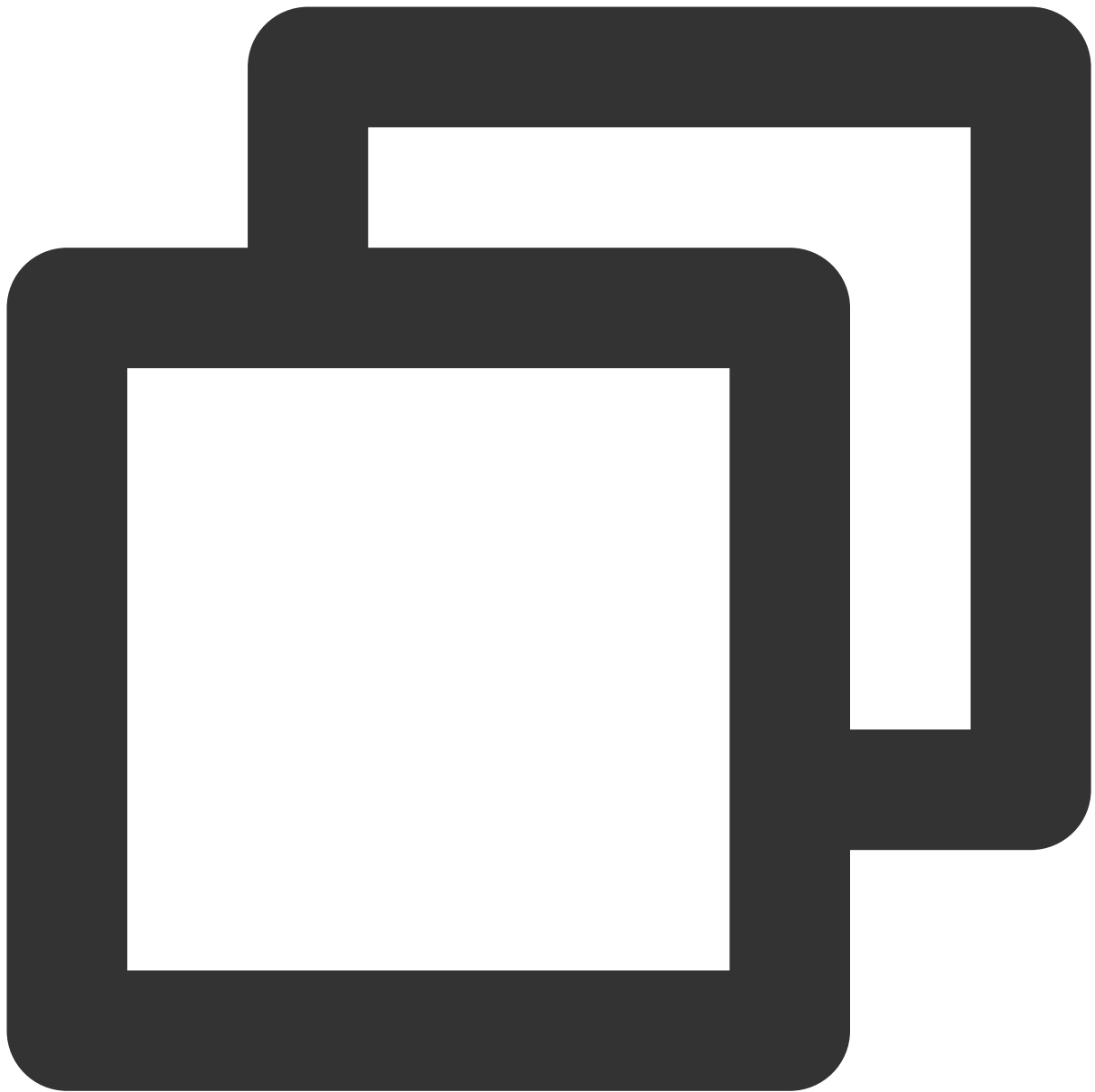
```
npm i -D sass sass-loader
```

Step 2. Download the TUIKit component

Download the TUIKit component through npm. To facilitate your subsequent expansion, it is recommended that you copy the TUIKit component to the src directory of your project:

macOS

Windows
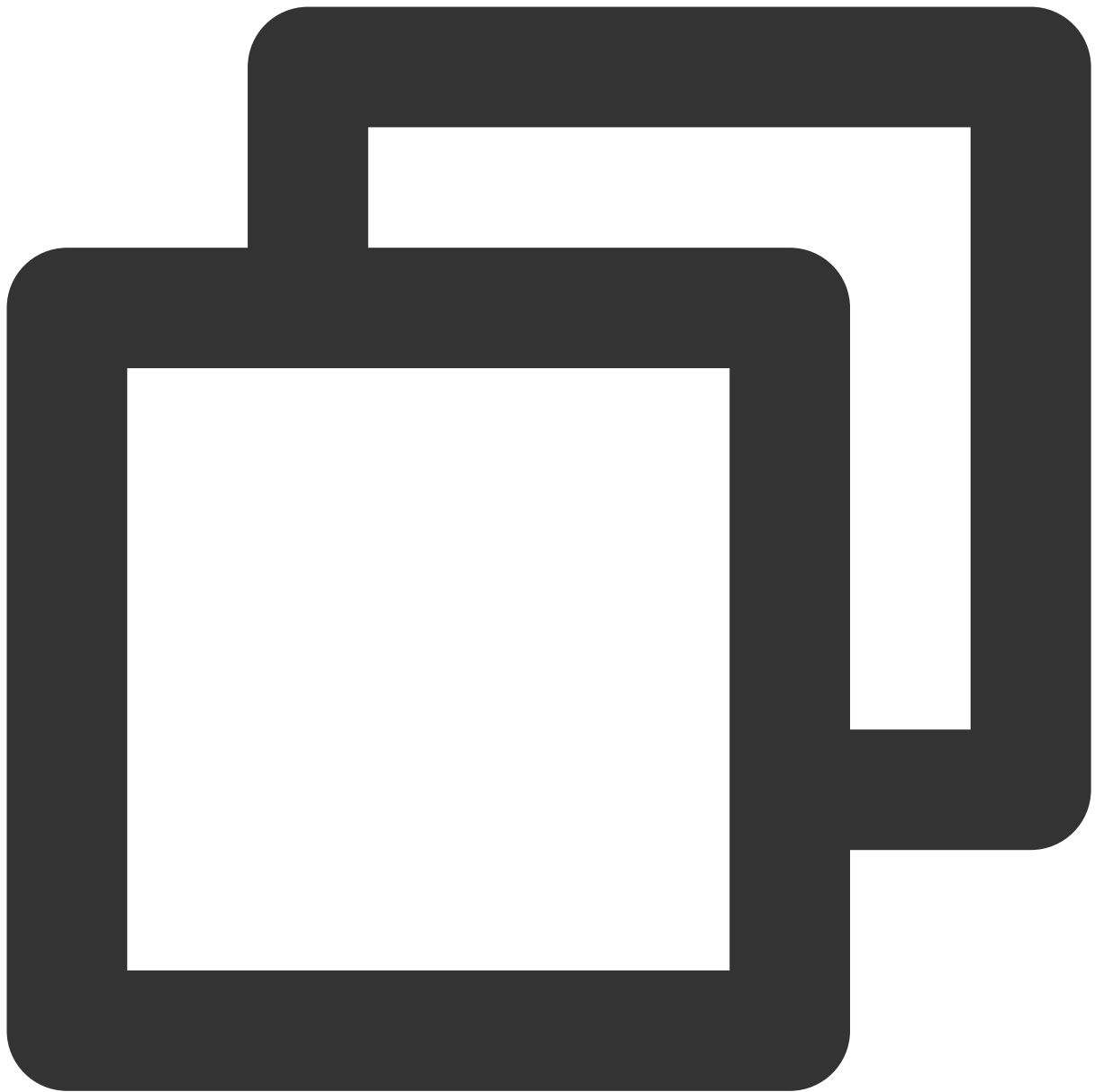
```
npm i @tencentcloud/chat-uikit-vue
mkdir -p ./src/TUIKit && rsync -av --exclude={'node_modules','package.json','exclud
```

```
npm i @tencentcloud/chat-uikit-vue
```
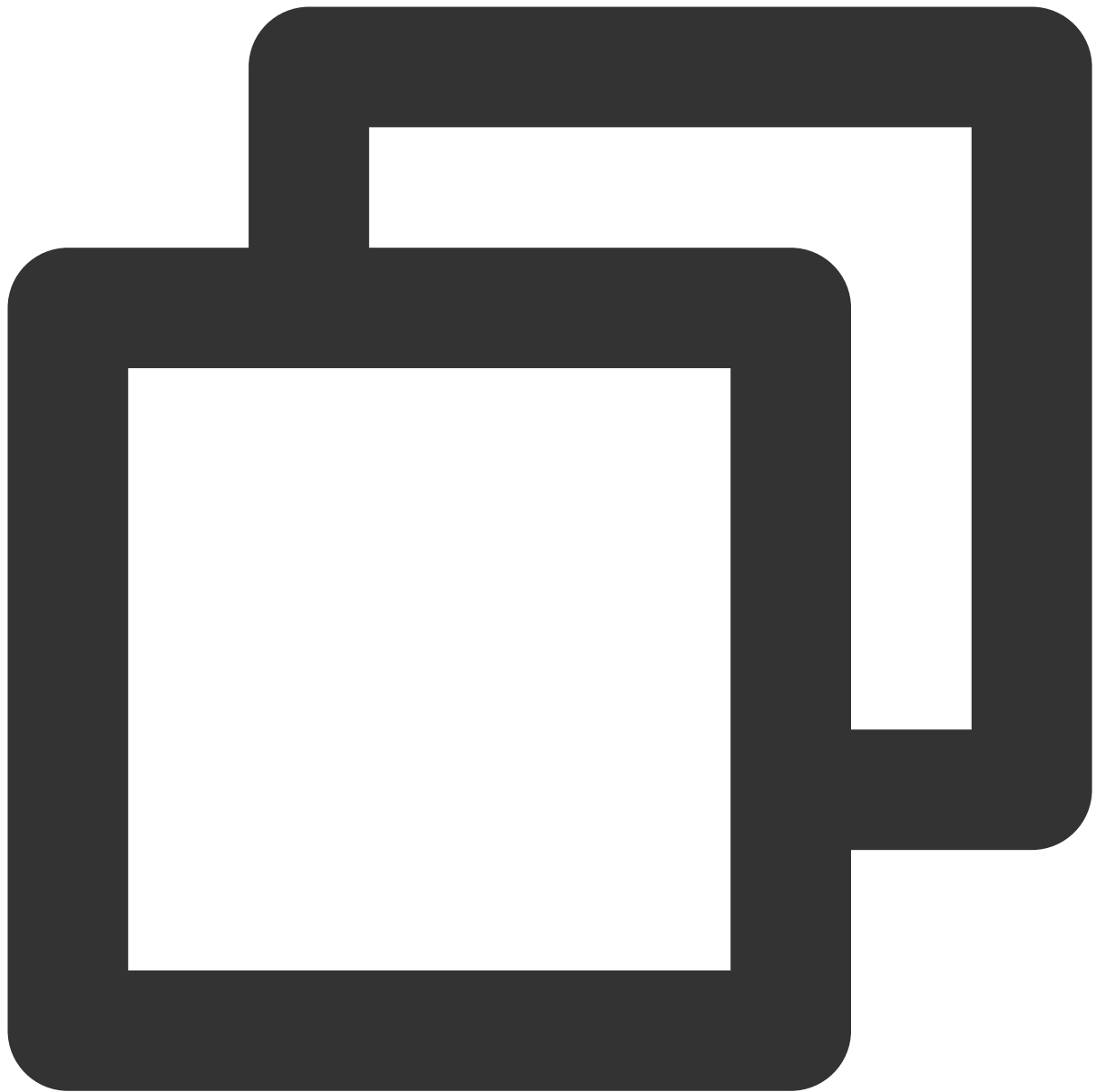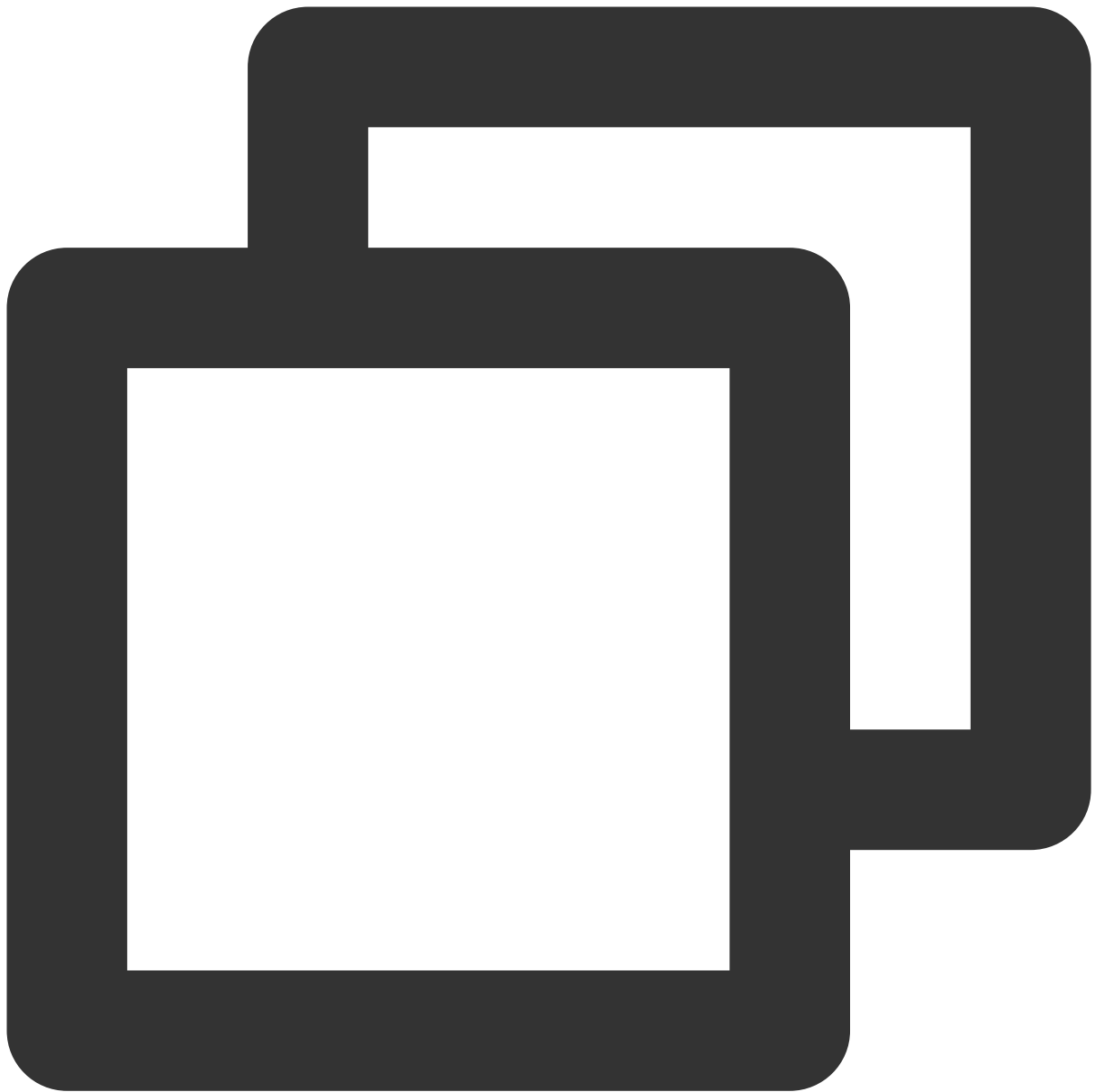
Tencent Cloud

```
xcopy .\\node_modules\\@tencentcloud\\chat-uikit-vue .\\src\\TUIKit /i /e /exclude:
```

Upon successful completion, the directory structure is depicted as follows:
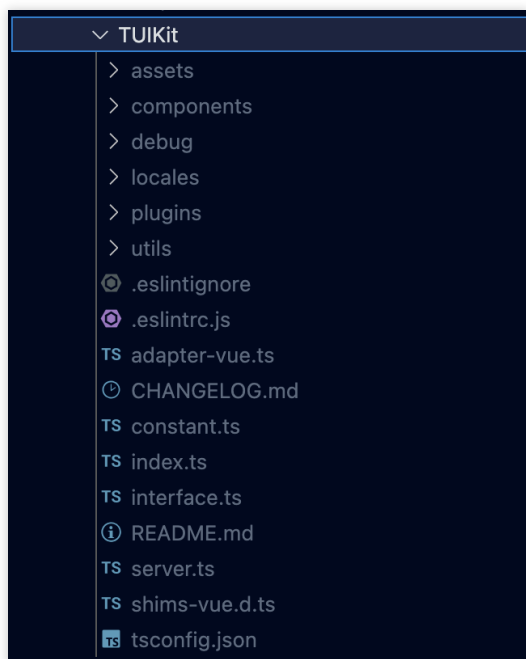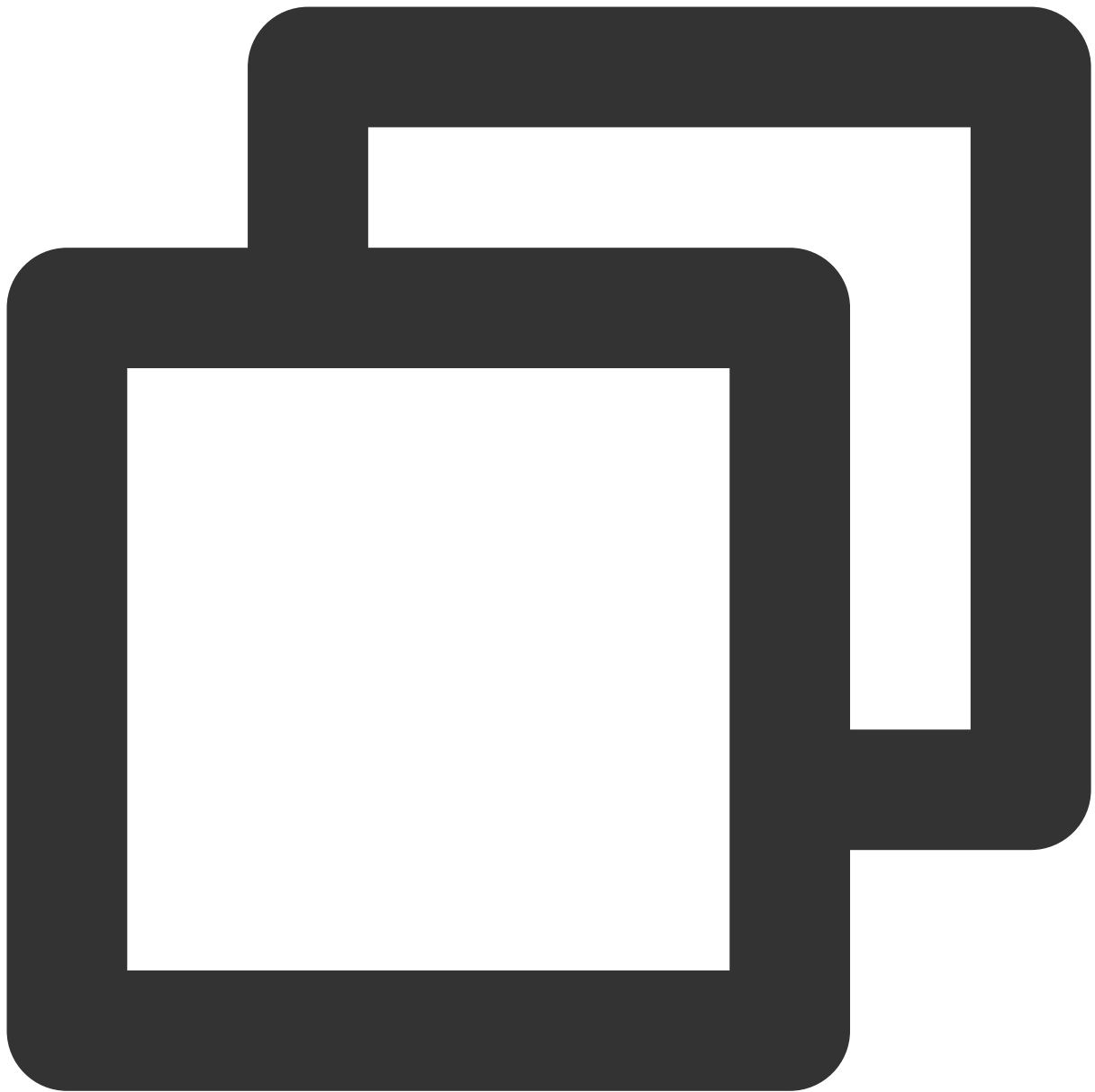
Step 3: Import TUIKit Component

3.1 Import TUIKit in `main.ts / main.js` **and register it into the Vue project instance**

Vue3

Vue2

```
import { createApp } from 'vue';
import App from './App.vue';
import { TUIComponents, TUIChatKit, genTestUserSig } from "./TUIKit";
import { TUILogin } from "@tencentcloud/tui-core";
const app = createApp(App);

const SDKAppID = 0; // Your SDKAppID
const secretKey = ""; //Your secretKey
const userID = ""; // User ID

// TUIChatKit add TUIComponents
```

```
TUIChatKit.components(TUIComponents, app);
// TUIChatKit init
TUIChatKit.init();
// TUICore login
TUILogin.login({
  SDKAppID,
  userID,
  // UserSig is a password used to log in to IM. It is the ciphertext obtained afte
  // this method is only suitable for locally running a demo project and feature de
  userSig: genTestUserSig({
    SDKAppID,
    secretKey,
    userID,
  }).userSig,
  useUploadPlugin: true,
  useProfanityFilterPlugin: false,
  framework: "vue3",
});

app.mount("#app");
export { SDKAppID, secretKey };
```
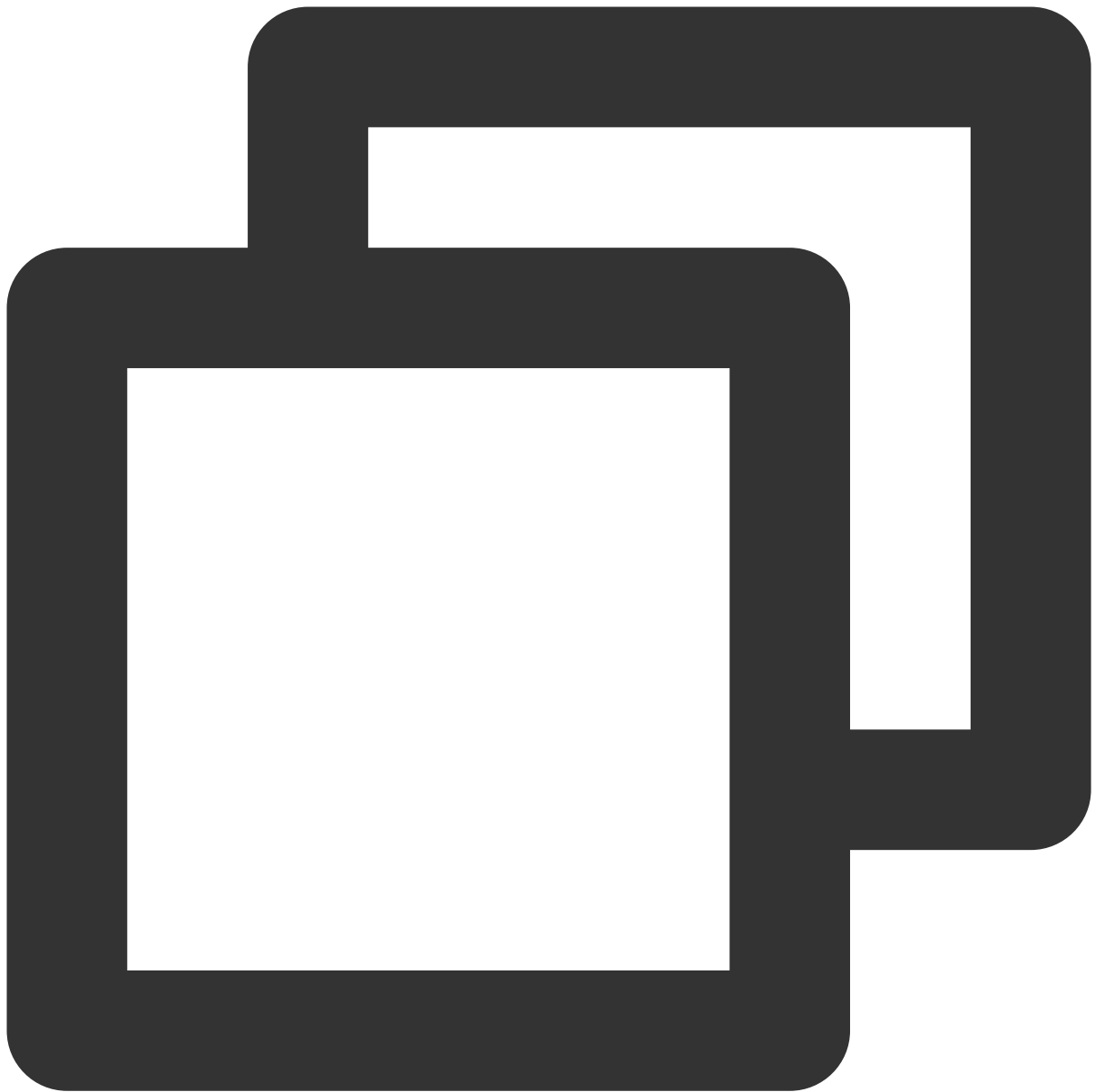
```
import Vue from "vue";
import App from "./App.vue";
import { TUIComponents, TUIChatKit, genTestUserSig } from "./TUIKit";
import { TUILogin } from "@tencentcloud/tui-core";

const SDKAppID = 0; // Your SDKAppID
const secretKey = ""; //Your secretKey
const userID = ""; // User ID

// TUIChatKit add TUIComponents
TUIChatKit.components(TUIComponents, Vue);
```

```
// TUIChatKit init
TUIChatKit.init();
// TUICore login
TUILogin.login({
  SDKAppID,
  userID,
  // UserSig is a password used to log in to IM. It is the ciphertext obtained afte
  // this method is only suitable for locally running a demo project and feature de
  userSig: genTestUserSig({
    SDKAppID,
    secretKey,
    userID,
  }).userSig,
  useUploadPlugin: true,
  // Local review can identify and handle unsafe and inappropriate content, safegua
  // This function is a value-added service, for reference please visit: https://cl
  // If you have purchased the content review service, please set this feature to `
  useProfanityFilterPlugin: false,
  framework: "vue2",
});
Vue.config.productionTip = false;
  new Vue({
    render: (h) => h(App),
  }).$mount("#app");
export { SDKAppID, secretKey };
```

3.2 Implement UIKit to your web app

Vue3

Vue 2.7

Vue2.6 or lower versions

For example: In the App.vue page, use TUIConversation, TUIChat, TUIContact, TUISearch, TUIGroup and TUICallKit
to quickly build a chat interface (the following sample code supports both the Web side and the H5 side).

Tencent Cloud

```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar = item.id"
      >
        {{ item.label }}
      </div>
```

```
        </div>
        <div class="TUIKit-main-container">
          <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
            <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
              <TUISearch searchType="global"></TUISearch>
              <TUIConversation></TUIConversation>
            </div>
            <div v-if="!isH5 || currentConversationID" class="TUIKit-main-main">
              <TUIChat>
                <h1>Welcome Chat</h1>
              </TUIChat>
              <TUIGroup :class="isH5 ? 'chat-popup' : 'chat-aside'" />
              <TUISearch :class="isH5 ? 'chat-popup' : 'chat-aside'" searchType="conver
            </div>
            <TUIGroup v-if="isH5 && !currentConversationID" class="chat-popup" />
            <TUIContact displayType="selectFriend" />
          </div>
          <div v-else-if="currentNavbar === 'contact'" class="TUIKit-main">
            <TUIContact
              displayType="contactList"
              @switchConversation="currentNavbar = 'conversation'"
            />
          </div>
          <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullSc
        </div>
      </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";
import { TUICallKit } from "@tencentcloud/call-uikit-vue";
import { TUISearch, TUIConversation, TUIChat, TUIContact, TUIGroup } from "./TUIKit
import { isH5 } from "./TUIKit/utils/env";
const currentConversationID = ref<string>("");
const currentNavbar = ref<string>("conversation");
const navbarList = [
  {
    id: "conversation",
    label: "Conversation",
  },
  {
    id: "contact",
    label: "Contact",
  },
];
TUIStore.watch(StoreName.CONV, {
  currentConversationID: (id: string) => {
```

```
        currentConversationID.value = id;
    },
});
</script>
<style scoped lang="scss">
@import "./TUIKit/assets/styles/common.scss";
@import "./TUIKit/assets/styles/sample.scss";
</style>
```

For example: In the App.vue page, use TUIConversation, TUIChat, TUIContact, TUISearch, TUIGroup and TUICallKit to quickly build a chat interface (the following sample code supports both the Web side and the H5 side).

Tencent Cloud

```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar = item.id"
      >
        {{ item.label }}
      </div>
```

```
          </div>
        <div class="TUIKit-main-container">
          <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
            <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
              <TUISearch searchType="global"></TUISearch>
              <TUIConversation></TUIConversation>
            </div>
            <div v-if="!isH5 || currentConversationID" class="TUIKit-main-main">
              <TUIChat>
                <h1>Let's Chat!</h1>
              </TUIChat>
              <TUIGroup :class="isH5 ? 'chat-popup' : 'chat-aside'" />
              <TUISearch :class="isH5 ? 'chat-popup' : 'chat-aside'" searchType="conver
            </div>
            <TUIGroup v-if="isH5 && !currentConversationID" class="chat-popup" />
            <TUIContact displayType="selectFriend" />
          </div>
          <div v-else-if="currentNavbar === 'contact'" class="TUIKit-main">
            <TUIContact
              displayType="contactList"
              @switchConversation="currentNavbar = 'conversation'"
            />
          </div>
          <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullSc
        </div>
      </div>
</template>
<script lang="ts">
import Vue from "vue";
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";
import { TUICallKit } from "@tencentcloud/call-uikit-vue2";
import { TUISearch, TUIConversation, TUIChat, TUIContact, TUIGroup } from "./TUIKit
import { isH5 } from "./TUIKit/utils/env";
export default Vue.extend({
  name: "App",
  components: {
    TUISearch,
    TUIGroup,
    TUIConversation,
    TUIChat,
    TUIContact,
    TUICallKit,
  },
  data() {
    return {
      isH5: isH5,
      currentConversationID: "",
```

```
      currentNavbar: "conversation",
      navbarList: [
        {
          id: "conversation",
          label: "Conversation",
        },
        {
          id: "contact",
          label: ",
        },
      ],
    };
  },
  mounted: function () {
    TUIStore.watch(StoreName.CONV, {
      currentConversationID: (id: string) => {
        this.currentConversationID = id;
      },
    });
  },
});
</script>
<style scoped lang="scss">
@import "./TUIKit/assets/styles/common.scss";
@import "./TUIKit/assets/styles/sample.scss";
</style>
```

1. Install the relevant dependencies supporting `composition-api` , `script setup` , and `vue2.6`

Tencent Cloud

```
npm i @vue/composition-api unplugin-vue2-script-setup vue@2.6.14 vue-template-compi
```

2. Import `VueCompositionAPI` in `main.ts`
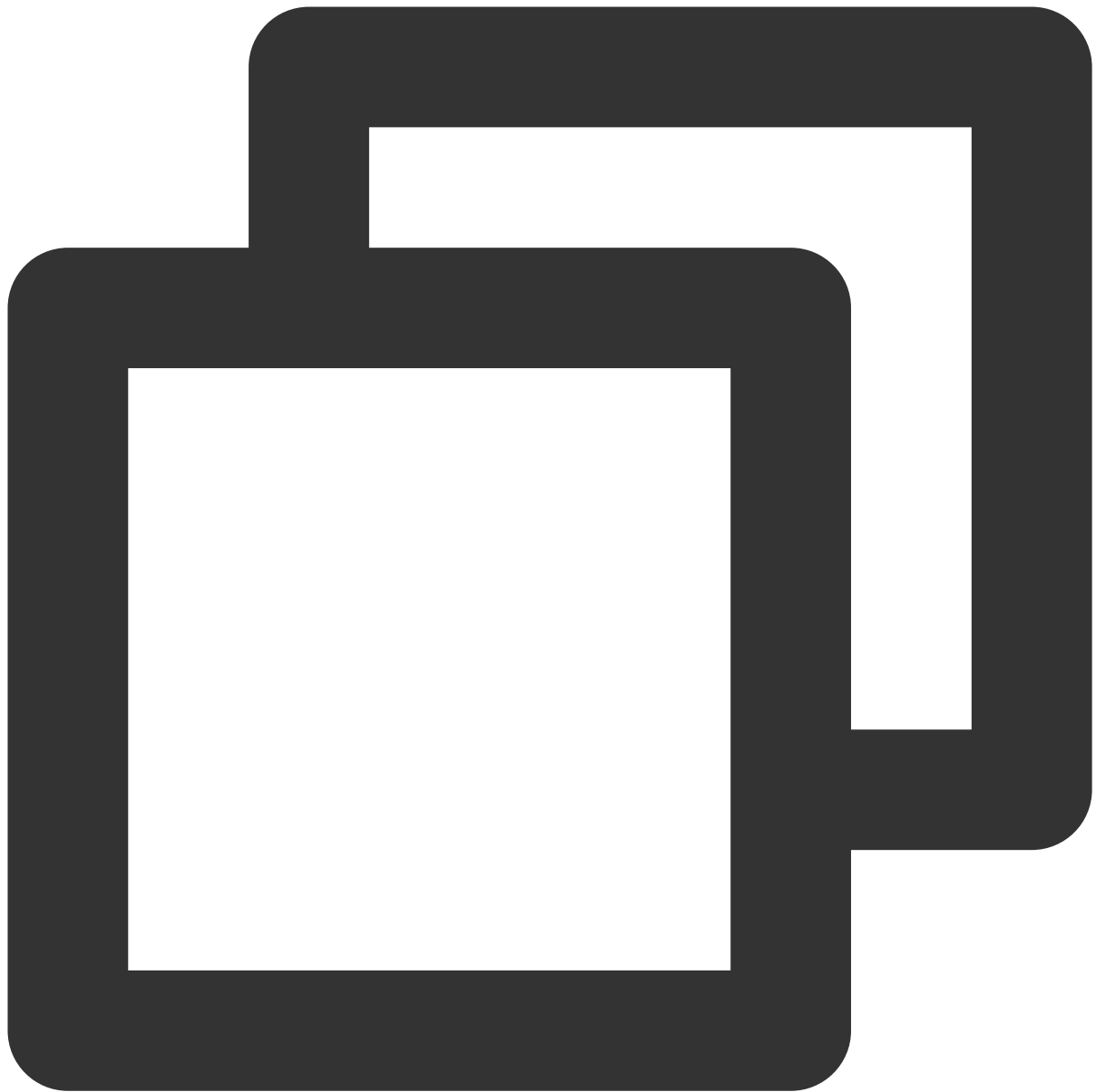
```
import VueCompositionAPI from "@vue/composition-api";
Vue.use(VueCompositionAPI);
```

3. Add the following pattern to `vue.config.js` , if the file doesn't exist, please create one:

```
const ScriptSetup = require("unplugin-vue2-script-setup/webpack").default;
module.exports = {
  parallel: false, // disable thread-loader, which is not compactible with this plu
  configureWebpack: {
    plugins: [
      ScriptSetup({
        /* options */
      }),
    ],
  },
  chainWebpack(config) {
```

```
    // disable type check and let `vue-tsc` handles it
    config.plugins.delete("fork-ts-checker");
  },
};
```

4. Replace the export source at the end of the `src/TUIKit/adapter-vue.ts` file:
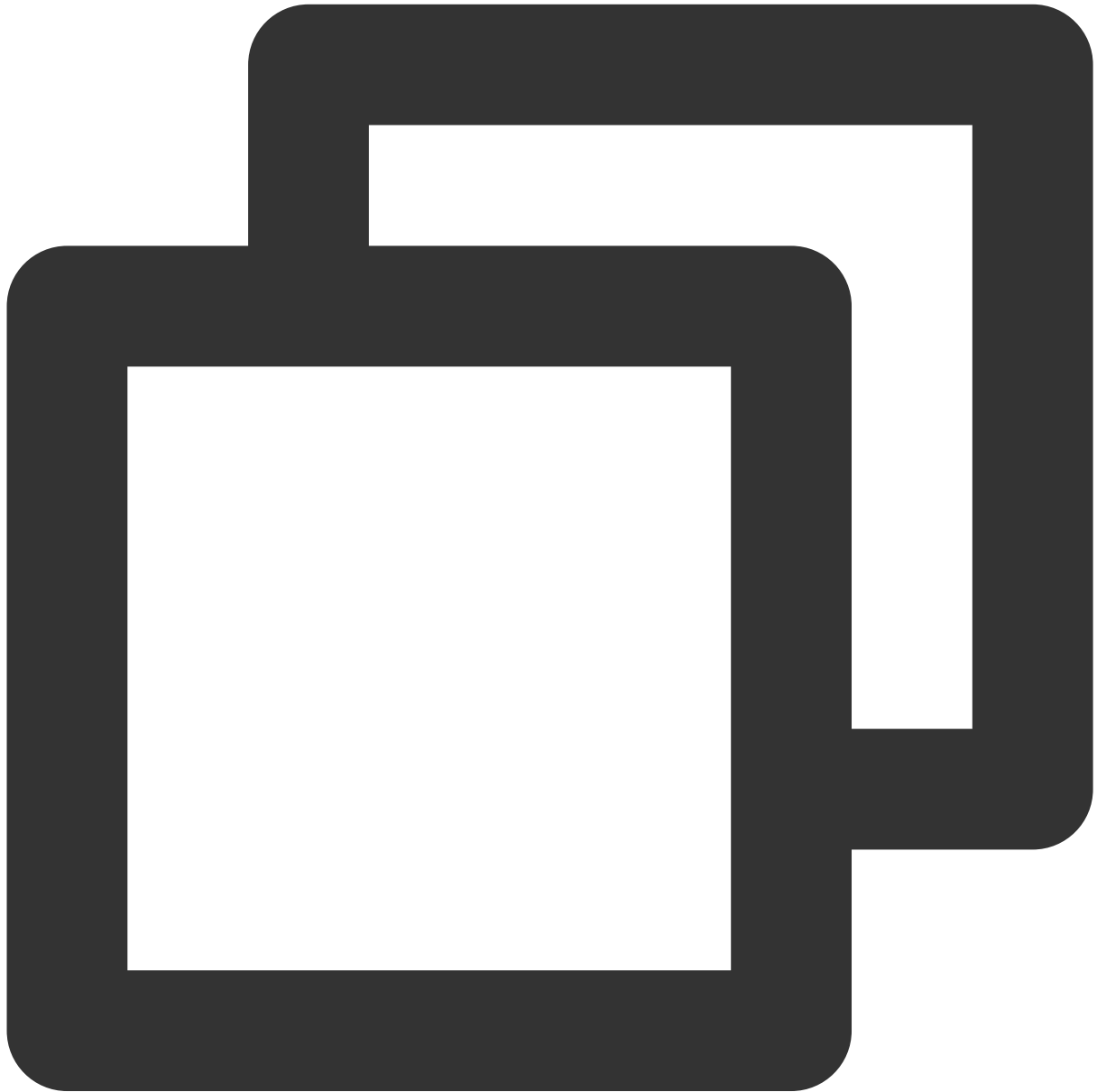

```
// Initial script
export * from "vue";
// Replace with
export * from "@vue/composition-api";
```

5. For example: In the App.vue page, use TUIConversation, TUIChat, TUIContact, TUISearch, TUIGroup and TUICallKit to quickly build a chat interface (the following sample code supports both the Web side and the H5 side).

```
<template>
  <div :class="['TUIKit', isH5 && 'TUIKit-h5']">
    <div v-if="!(isH5 && currentConversationID)" class="TUIKit-navbar">
      <div
        v-for="item of navbarList"
        :key="item.id"
        :class="['TUIKit-navbar-item', currentNavbar === item.id && 'TUIKit-navbar-
        @click="currentNavbar = item.id"
```

```
      >
        {{ item.label }}
      </div>
    </div>
    <div class="TUIKit-main-container">
      <div v-if="currentNavbar === 'conversation'" class="TUIKit-main">
        <div v-if="!(isH5 && currentConversationID)" class="TUIKit-main-aside">
          <TUISearch searchType="global"></TUISearch>
          <TUIConversation></TUIConversation>
        </div>
        <div v-if="!isH5 || currentConversationID" class="TUIKit-main-main">
          <TUIChat>
            <h1>Let's Chat!</h1>
          </TUIChat>
          <TUIGroup :class="isH5 ? 'chat-popup' : 'chat-aside'" />
          <TUISearch :class="isH5 ? 'chat-popup' : 'chat-aside'" searchType="conver
        </div>
        <TUIGroup v-if="isH5 && !currentConversationID" class="chat-popup" />
        <TUIContact displayType="selectFriend" />
      </div>
      <div v-else-if="currentNavbar === 'contact'" class="TUIKit-main">
        <TUIContact
          displayType="contactList"
          @switchConversation="currentNavbar = 'conversation'"
        />
      </div>
      <TUICallKit class="callkit-container" :allowedMinimized="true" :allowedFullSc
    </div>
  </div>
</template>
<script lang="ts">
import Vue from "vue";
import { TUIStore, StoreName } from "@tencentcloud/chat-uikit-engine";
import { TUICallKit } from "@tencentcloud/call-uikit-vue2.6";
import { TUISearch, TUIConversation, TUIChat, TUIContact, TUIGroup } from "./TUIKit
import { isH5 } from "./TUIKit/utils/env";
export default Vue.extend({
  name: "App",
  components: {
    TUISearch,
    TUIGroup,
    TUIConversation,
    TUIChat,
    TUIContact,
    TUICallKit,
  },
  data() {
```

```
    return {
      isH5: isH5,
      currentConversationID: "",
      currentNavbar: "conversation",
      navbarList: [
        {
          id: "conversation",
          label: "Conversation",
        },
        {
          id: "contact",
          label: "Contact",
        },
      ],
    };
  },
  mounted: function () {
    TUIStore.watch(StoreName.CONV, {
      currentConversationID: (id: string) => {
        this.currentConversationID = id;
      },
    });
  },
});
</script>
<style scoped lang="scss">
@import "./TUIKit/assets/styles/common.scss";
@import "./TUIKit/assets/styles/sample.scss";
</style>
```

Step 4: Secure SDKAppID, secretKey, and userID

Set the relevant parameters `SDKAppID` , `secretKey` , and `userID` in the example code of the `main.ts /`
`main.js` file:

`SDKAppID` and `SecretKey` can be accessed by the Instant Messaging console:

`UserID` can be accessed by the Instant Messaging Console > Account Management . Switch to the target application account to create an account and get the userID.
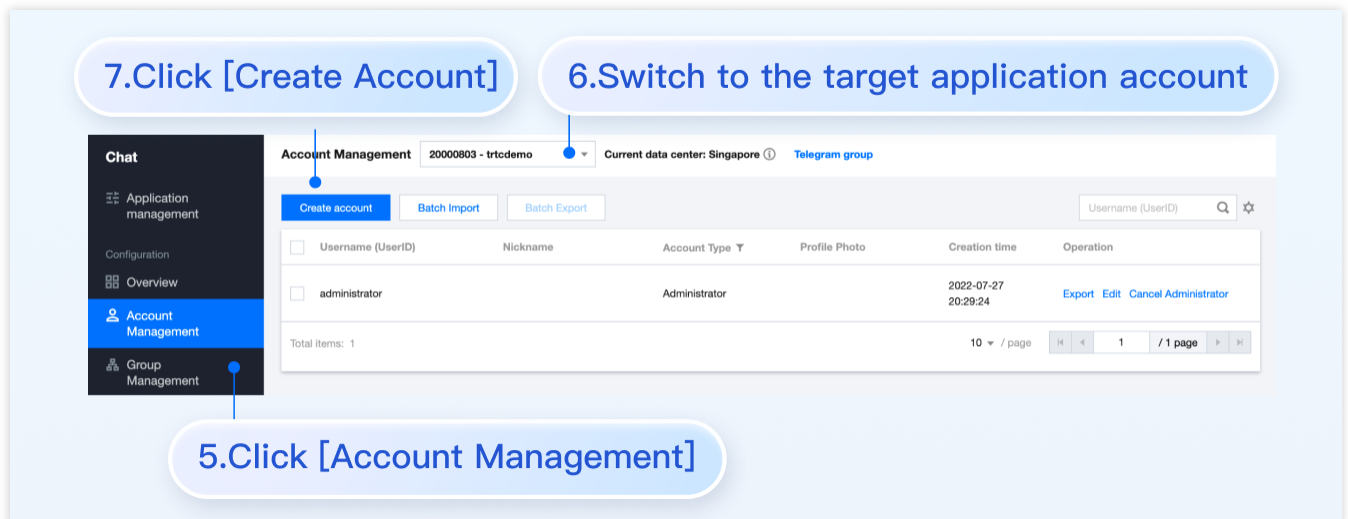
Step 5. Launch the project

vue-cli

vite
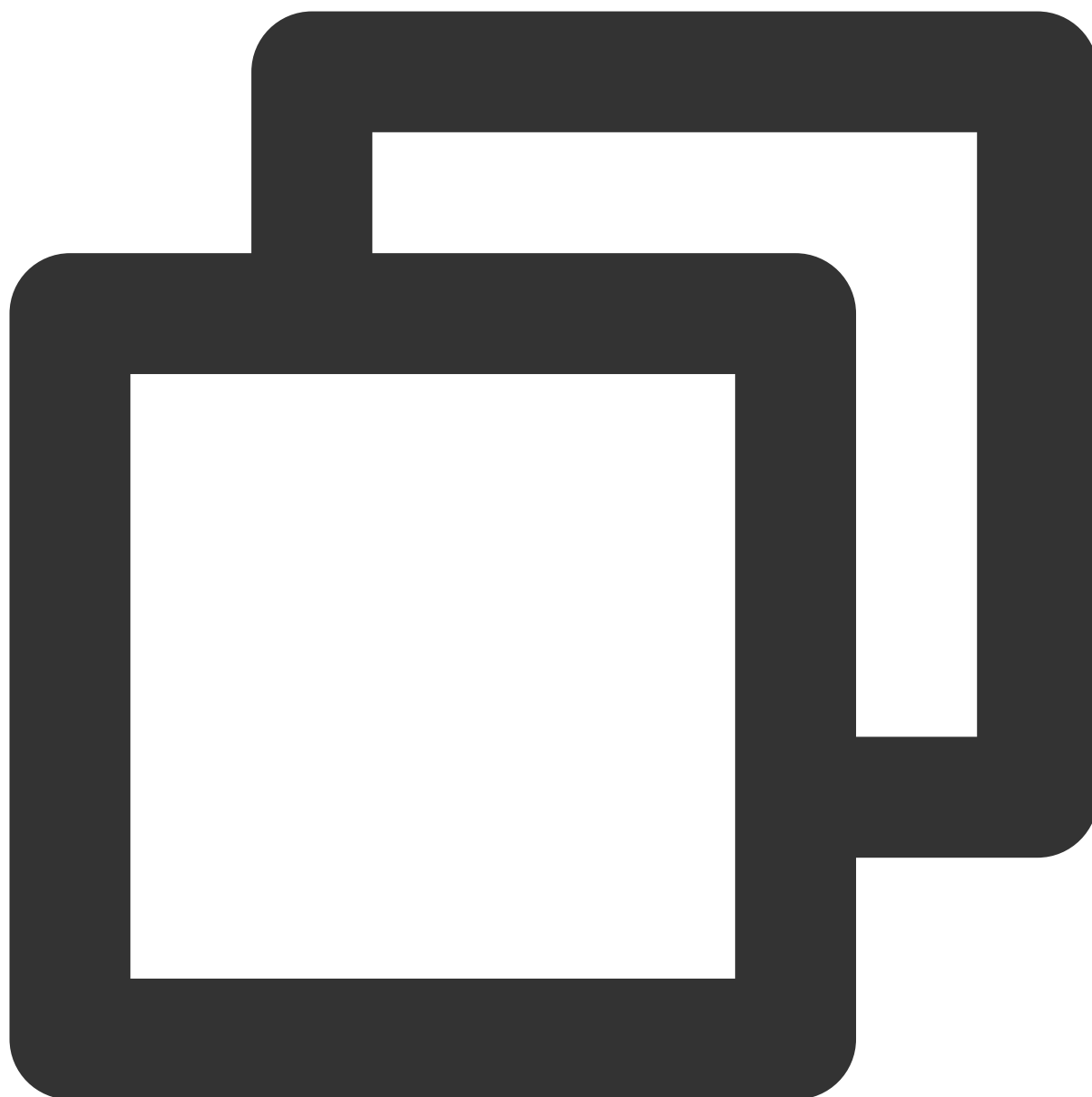
```
npm run serve
```

```
npm run dev
```

Additional item: Switching languages

The `Vue TUIKit` comes with default **Simplified Chinese, English** language packages that serve as the interface display language.
You may switch languages through the following methods.

```
import { TUITranslateService } from "@tencentcloud/chat-uikit-engine";
// change language to chinese
TUITranslateService.changeLanguage("zh");
// change language to english
TUITranslateService.changeLanguage("en");
```

Step 6. Send your first message

Tencent Cloud

Conversation

Contact

Search

aaa
Good morning. Have a wonderful day.   Just now

aaa

【Security Tips】This APP is only used to experience the functions of Tencent Cloud Instant Messaging products, and cannot be used for business negotiation as remittances and lottery wins, don't make unfamiliar phone calls easily, and beware of being deceived.

20:42

Step1. New one-to-one chat

Step2. Enter the userID you created in "Step 4" and press the "enter" key to search

1111

aaa

New one-to-one chat

Cancel   Done

Step3. Select the target user and click "Done" to new the chat

Enter a message

Step

Step1. New one–to–one chat

Step2. Enter the userID you created in "Step 4" and press the "enter" key to search

Step3. Select the target user and click "Done" to new the chat

Step4. Send

Step 7: Make your first phone call
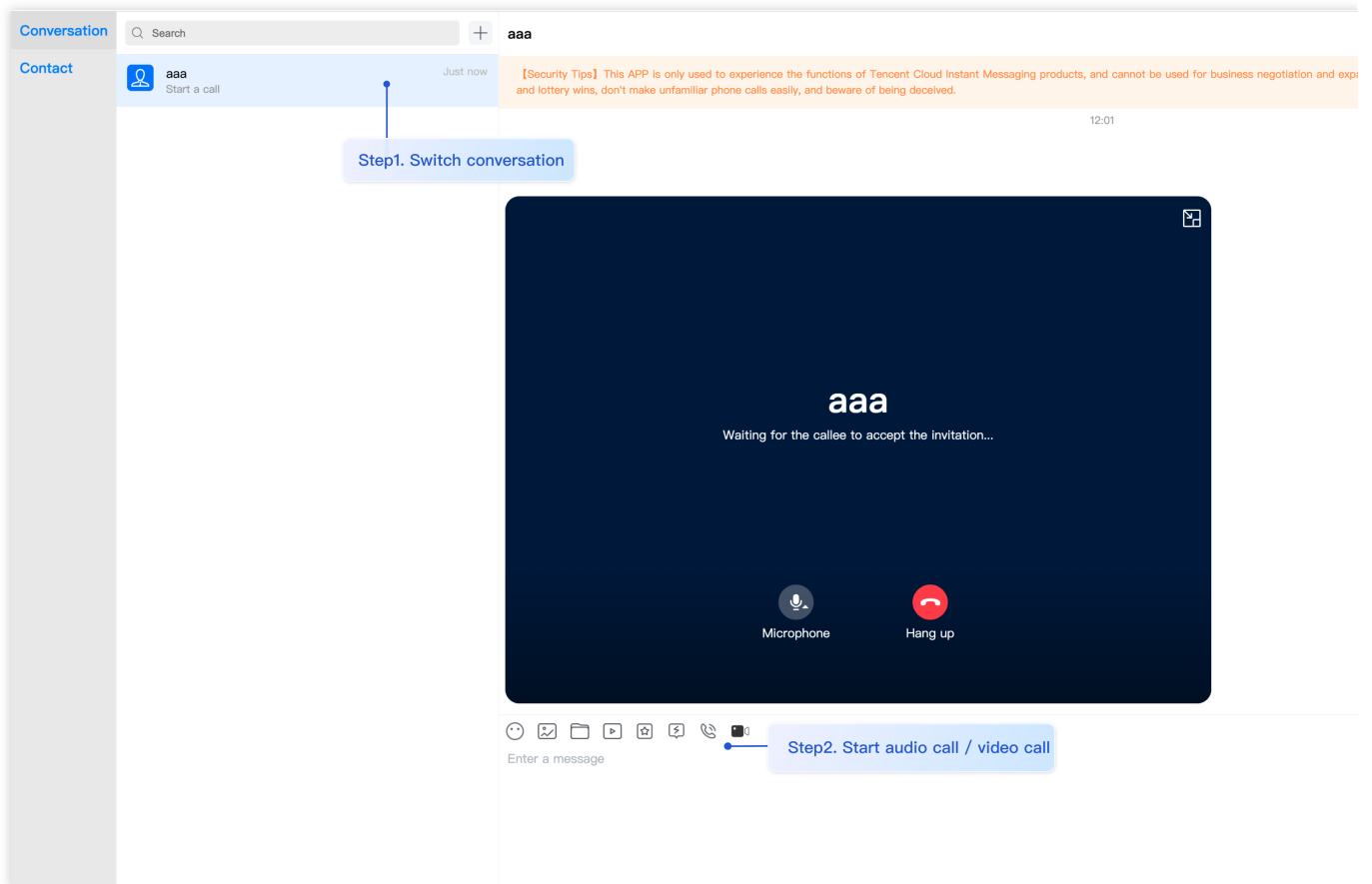
FAQs

Product Service FAQs

1. The Audio/Video Call Capability package is not activated? Failure to initiate the Audio/Video Call?

Please click Audio/Video Call > Frequently Asked Questions to view the solutions.

2. What is UserSig? How is UserSig generated?

A UserSig is a password with which you can log in to use IM service. It is the ciphertext generated by encrypting information such as userID.

The issuance of UserSig is achieved by integrating the calculation code for UserSig into your server-side, whilst providing an interface designed for your project. Whenever UserSig is required, your project could request the operational server for a dynamic UserSig. For further information, please refer to Generating UserSig on the server-side.

Caution

The method to obtain UserSig demonstrated in this document utilizes the configuration of a SECRETKEY within the client-side code. Within this procedure, the SECRETKEY is notably vulnerable to decompilation and reverse-

engineering. Should your SECRETKEY be leaked, malefactors could potentially exploit your Tencent Cloud traffic. Therefore, **this technique is only appropriate for local operation and functional debugging**. For the correct method of issuing UserSig, please refer to the earlier text.

Connection Errors FAQs

1. Runtime error: "TypeError: Cannot read properties of undefined (reading "getFriendList")"

If the following errors occur during runtime after connecting as per the steps outlined above, it is imperative that you **delete the node_modules directory under the TUIKit folder** to ensure the uniqueness of TUIKit's dependencies, preventing issues caused by multiple copies of dependencies.

2. How does a JS project integrate the TUIKit component?

TUIKit exclusively supports the TS environment for operation. You can enable the coexistence of existing JS code in your project with the TS code in TUIKit through progressive configuration of TypeScript.

vue-cli

vite

Please execute the following in the root directory of your engineering project created by the Vue CLI scaffold:

```
vue add typescript
```

Subsequently, please make selections in accordance with the following configuration options. To assure that we can support both the existing js code and the ts code within TUIKit, it is imperative that you strictly adhere to the five options presented below.

```
Run `npm audit` for details.
✔  Successfully installed plugin: @vue/cli-plugin-typescript

? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpi
? Convert all .js files to .ts? No
? Allow .js files to be compiled? Yes
? Skip type checking of all declaration files (recommended for apps)? Yes


🚀   Invoking generator for @vue/cli-plugin-typescript...
📦   Installing additional dependencies...
```

Once these steps are completed, please rerun the project!

Please execute the following command in your project's root directory created with vite:

```
npm install -D typescript
```

3. Runtime error reported: /chat-example/src/TUIKit/components/TUIChat/message-input/message-input-editor.vue .ts(8,23)TS1005: expected.

```
98% after emitting CopyPlugin
 ERROR   Failed to compile with 2 errors                        16:20:59
 error   in                              /chat-example/src/TUIKit/components/TUIChat/message-input/message-input-edit
[tsl] ERROR in                              /chat-example/src/TUIKit/components/TUIChat/message-input/message-input-
3)
     TS1005: ',' expected.
```

The above error message appears because your installed @vue/cli version is too low. **You must ensure that your @vue/cli version is 5.0.0 or higher**. The upgrade method is as follows:
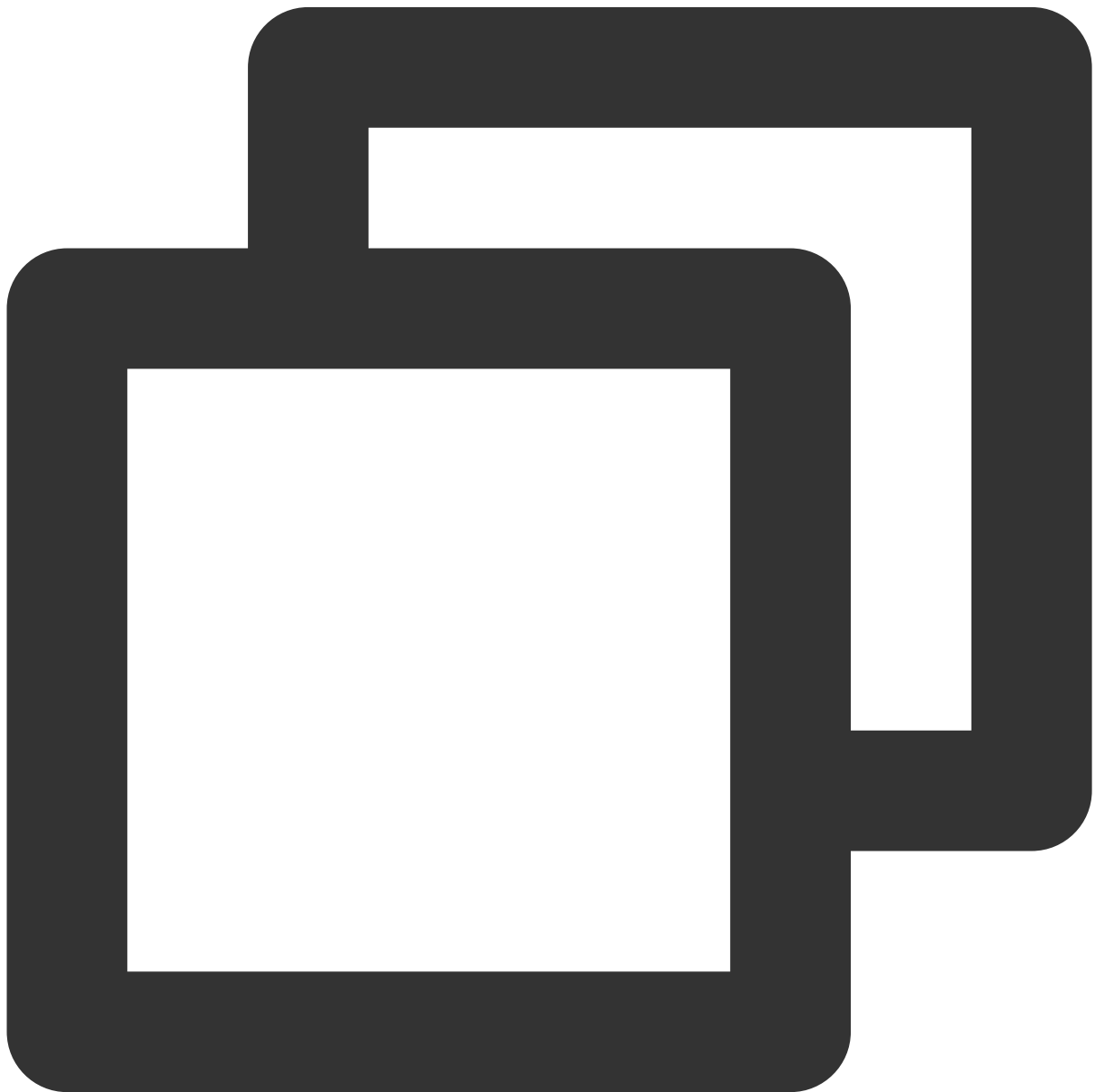

```
npm install -g @vue/cli@5.0.8
```

4. Runtime error: Failed to resolve loader: sass-loader

The above error message appears because the `sass` environment is not installed on your machine, please run the following command to install the `sass` environment:

```
npm i -D sass sass-loader@10.1.1
```

5. Other ESLint errors?

If copying chat-uikit-vue to the src directory results in error due to inconsistency with your local project code style, you may ignore this component directory. This can be achieved by adding .eslintignore file to the project root directory:

```
# .eslintignore
src/TUIKit
```

6. How to disable the full screen overlay error message prompt of webpack in dev mode in vue/cli?

You can disable it in the vue.config.js file at the root directory of your project:

webpack4

webpack3

```
module.exports = defineConfig({
  ...
  devServer: {
    client: {
      overlay: false,
    },
  },
});
```

```
module.exports = {
  ...
  devServer: {
    overlay: false,
  },
};
```

7. What to do when encountering 'Component name "XXXX" should always be multi-word'?

The version of ESLint utilized in IM TUIKit web is v6.7.2, which does not rigidly verify the camelCase format for module names.

Should you encounter this dilemma, you may configure as follows in the .eslintrc.js file:
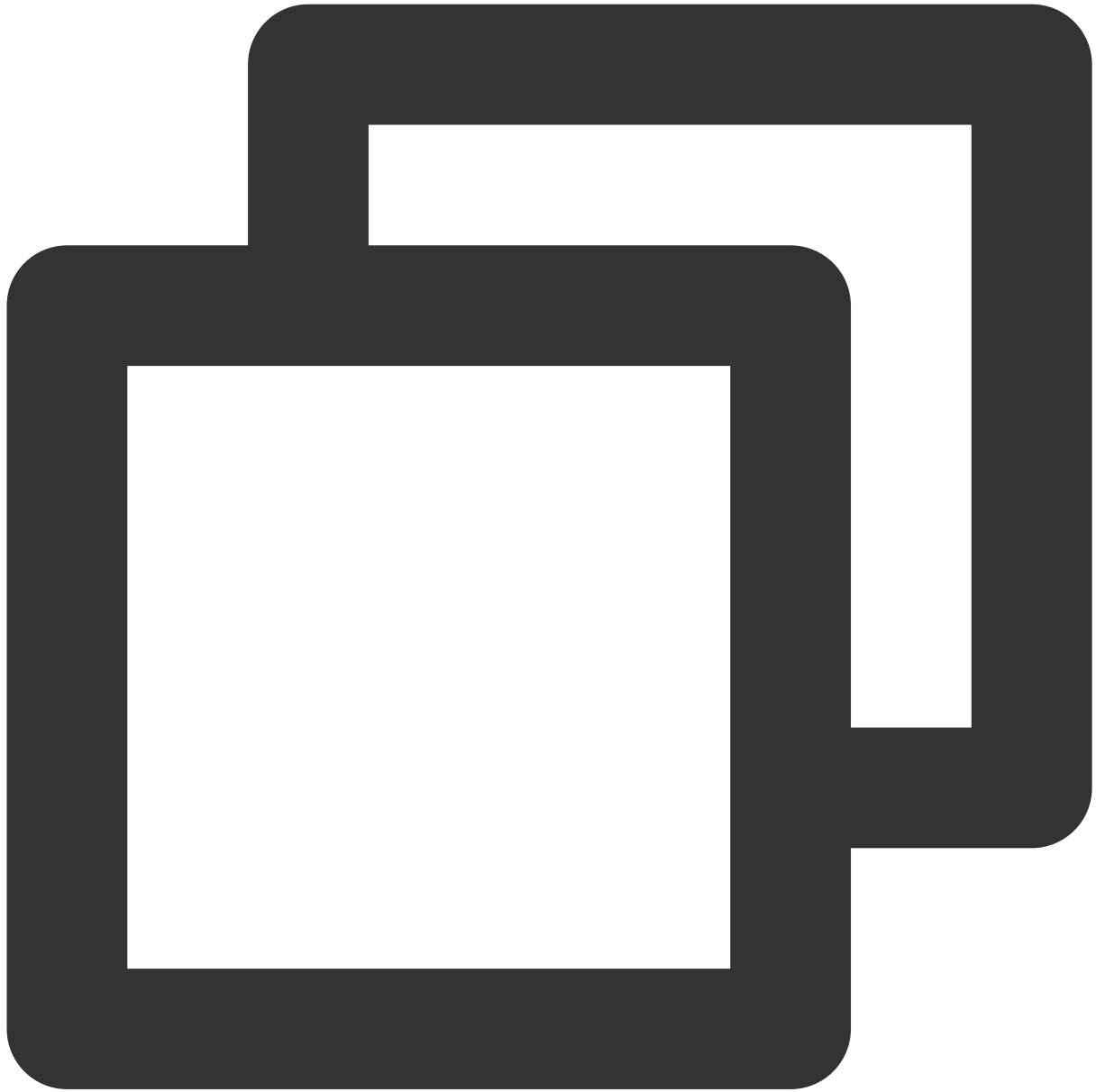

```
module.exports = {
  ...
  rules: {
     ...
     'vue/multi-word-component-names': 'warn',
  },
};
```

8. What should I do if I encounter ERESOLVE unable to resolve dependency tree?

If ERESOLVE unable to resolve dependency tree appears when npm install is run, it indicates a conflict in dependency installation. The following method can be adopted for installation:
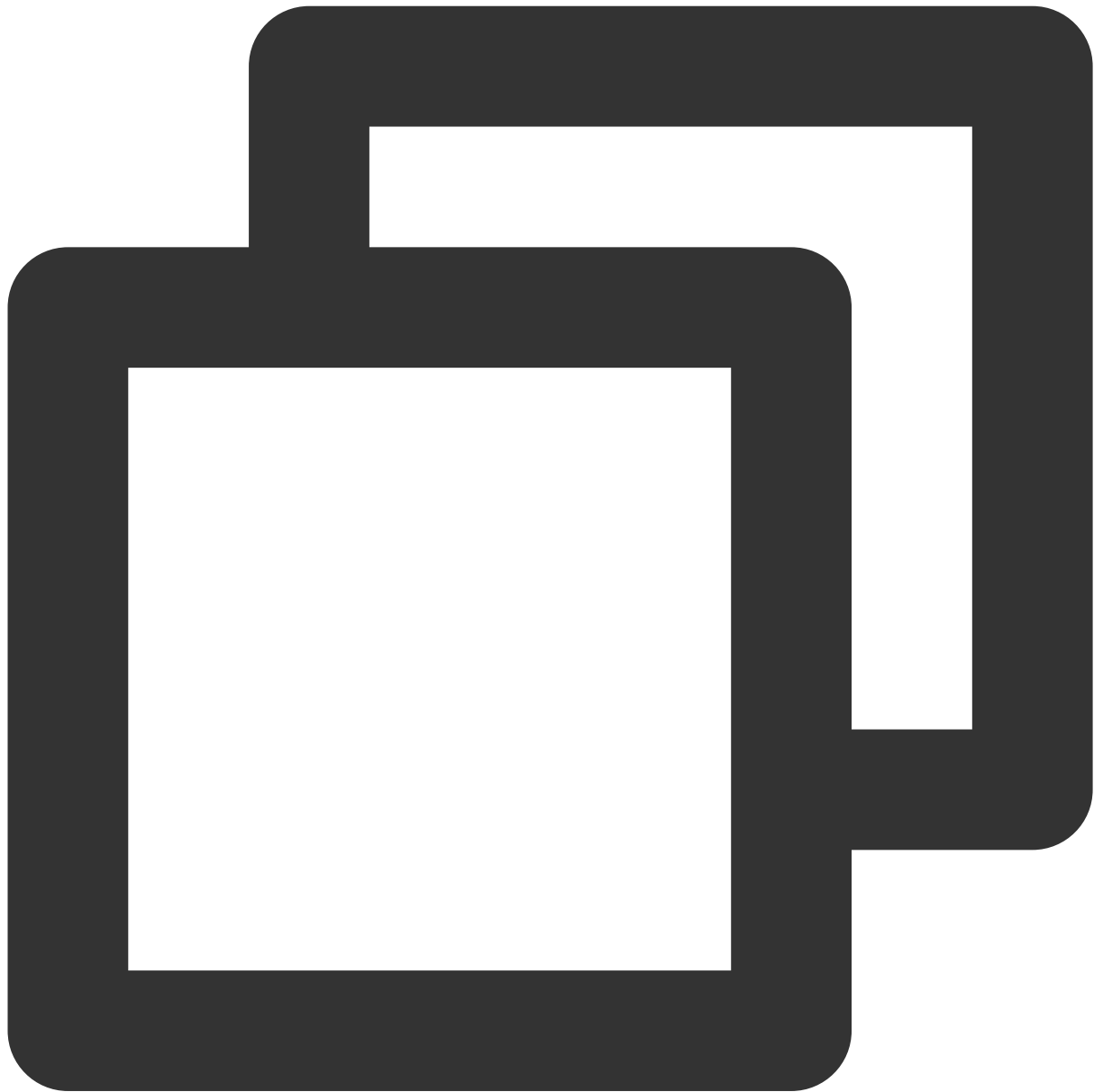

```
npm install --legacy-peer-deps
```

9. How might one address the error message, 'vue packages version mismatch' occurring during execution?
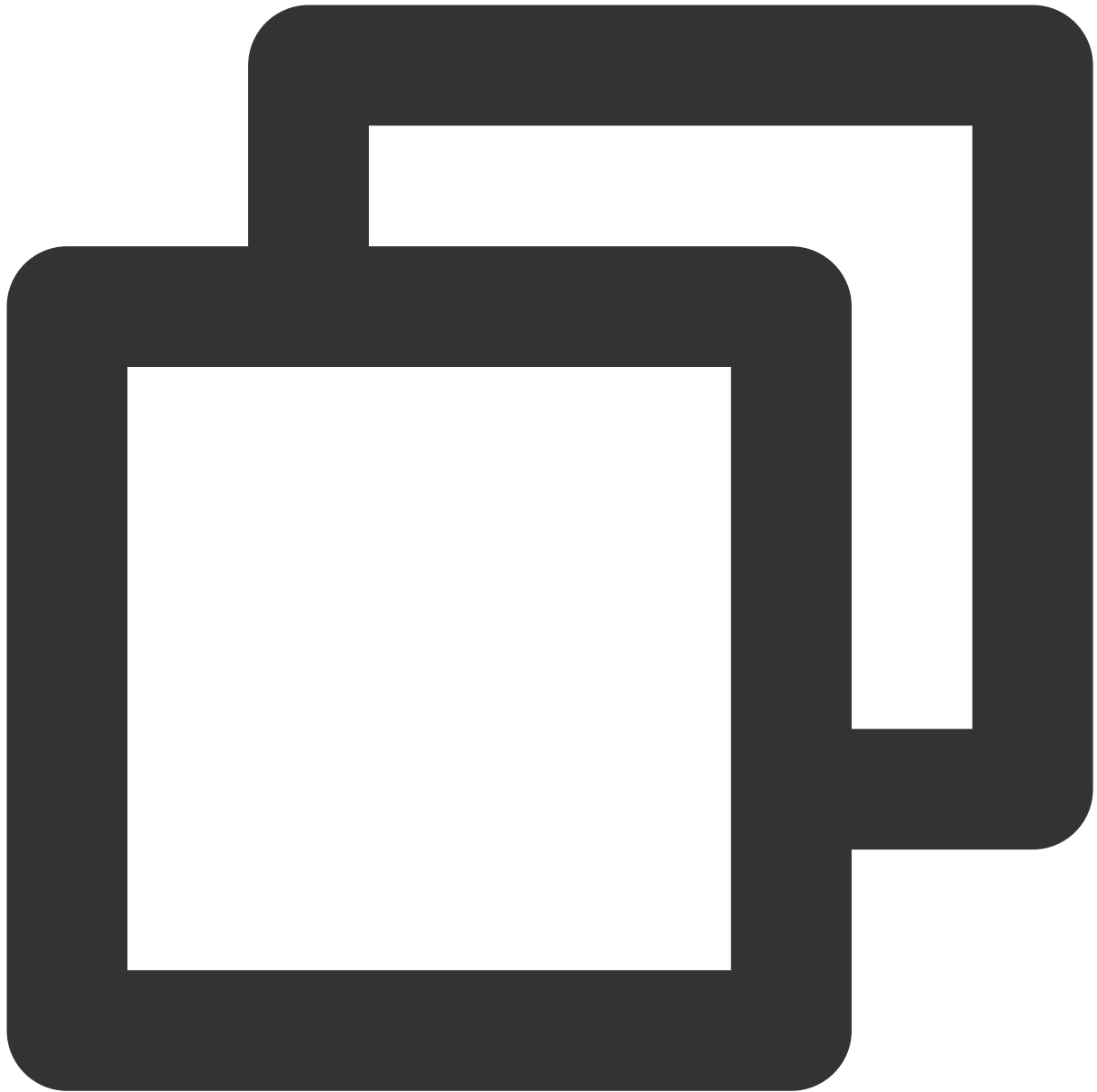
```
// If you are using a vue2.7 project, please execute in your project root directory
npm i vue@2.7.9 vue-template-compiler@2.7.9
// If you have a Vue2.6 project, please execute in your project's root directory
npm i vue@2.6.14 vue-template-compiler@2.6.14
```

10. Why does TypeScript report an error after npm run build in a Vite project?

```
node_modules/@tencentcloud/tui-customer-service-plugin/components/message-rating/index.vue:3:35 - error TS2551: Property 'RATING_TEMPLATE_TYPE' does not exist on type
ops: Partial<{}> & Omit<{ readonly message?: Record<string, any>; onSendMessage?: (...args: any[]) => any; } & ... 4 more ... & { ...; }, never>; ... 10 more ...; $wat
source: T, cb: T extends (...args: any) => infer R ? (arg...'. Did you mean 'ratingTemplate'?

3        v-if="ratingTemplate.type === RATING_TEMPLATE_TYPE.STAR"
                 ~~~~~~~~~~~~~~~~~~~~

src/TUIKit/components/common/FetchMore/index.vue:66:16 - error TS2304: Cannot find name 'uni'.

66      observer = uni
                   ~~~

src/TUIKit/components/common/ImagePreviewer/index.vue:164:7 - error TS2322: Type 'Timeout' is not assignable to type 'number'.

164     timer = setTimeout(() => {
            ~~~~~

src/TUIKit/components/common/ImagePreviewer/index.vue:189:5 - error TS2322: Type 'Timeout' is not assignable to type 'number'.

189     timer = setTimeout(() => {
            ~~~~~

src/TUIKit/components/common/Transfer/index.vue:97:21 - error TS2365: Operator '>' cannot be applied to types '{ toString: (radix?: number) => string; toFixed: (fracti
ractionDigits?: number) => string; toPrecision: (precision?: number) => string; valueOf: () => number; toLocaleString: { ...; }; }' and 'number'.

97              v-if="transferTotal > transferList.length"
                      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

src/TUIKit/components/TUIChat/chat-header/index.vue:17:39 - error TS2345: Argument of type 'string | number | object | { onClicked?: Function; onLongPressed?: Function
 is not assignable to parameter of type 'ExtensionInfo'.
  Type 'string' is not assignable to type 'ExtensionInfo'.

17          @click.stop="handleExtensions(item)">
                         ~~~~~~~~~~~~~~~~~~~~~

src/TUIKit/components/TUIChat/chat-header/index.vue:18:27 - error TS2339: Property 'icon' does not exist on type 'string | number | object | { onClicked?: Function; on
nSwiped?: Function; }'.
  Property 'icon' does not exist on type 'string'.

18          <Icon :file="item.icon"></Icon>
                        ~~~~~~~~~

src/TUIKit/components/TUIChat/chat-header/index.vue:41:39 - error TS2345: Argument of type 'undefined[]' is not assignable to parameter of type 'ExtensionInfo'.
  Type 'undefined[]' is missing the following properties from type 'ExtensionInfo': weight, text, icon, data, listener

41 const extensions = ref<ExtensionInfo>([])
                       ~~~~~~~~~~~~~~~~~~~~~
```

Reason: It's led by the vue-tsc command in "build": "vue-tsc && vite build" under package.json script.

Solution: Simply remove vue-tsc. "build": "vite build"

```
"scripts": {
  "dev": "vite",
  'build': "vite build",
  "preview": "vite preview"
},
```

Contact Us

Join the Telegram technical discussion group or WhatsApp discussion group, enjoy the support of professional engineers, and solve your difficulties.

Documentation

Related to Vue2 & Vue3 UIKit:

chat-uikit-vue npm
Vue2 Demo Source Code and Running Example
Vue3 Demo Source Code and Running Example

Vue2 & Vue3 UIKit logic layer: engine

chat-uikit-engine npm
chat-uikit-engine interface

Unity

Last updated：2024-04-23 15:52:34

This document describes how to integrate the SDK for Unity.

Environment Requirements

Environment	Version
Unity	2019.4.15f1 or later
Android	Android Studio 3.5 or later; devices with Android 4.1 or later for apps
iOS	Xcode 11.0 or later. Ensure that your project has a valid developer signature.

Supported Platforms

We are committed to building a set of Chat SDK and TUIKit for all Unity platforms, allowing you to run one set of code on all platforms.

Platform	Chat SDK
iOS	Supported
Android	Supported
macOS	Supported
Windows	Supported
Web	Supported from 1.8.1+

Note

For web, you need to perform a few extra steps for SDK integration. For details, see Part 5.

Prerequisites

1、You have signed up for a Tencent Cloud account and completed identity verification.

2、You have created an application as instructed in Creating and Upgrading an Application and recorded the SDKAppID.

Part 1. Creating Test Accounts

In the Chat console, select your application and click **Auxiliary Tools** > **UserSig Generation & Verification** on the left sidebar. Create two `UserID` values and their `UserSig` values, and copy the `UserID` , `Key` , and `UserSig` for subsequent logins.

Note

The test account is for development and testing only. Before the application is launched, the correct method for generating a UserSig is to integrate the UserSig calculation code into your server and provide an application-oriented API. When UserSig is needed, your application can send a request to the business server for a dynamic UserSig. For more information, see Generating UserSig.
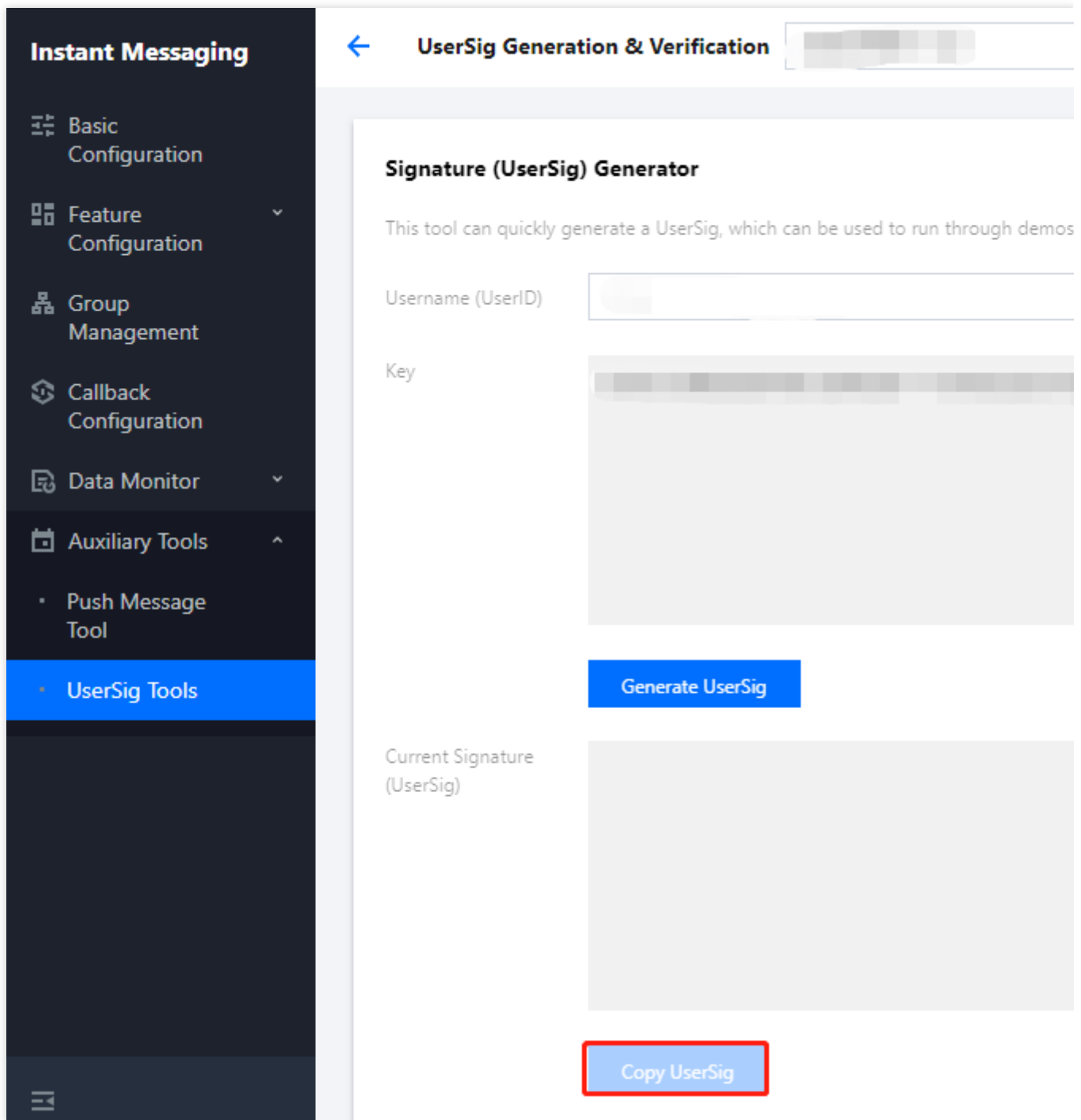
Part 2. Integrating Chat SDK into Your Unity Project

1.1 Use Unity to create a project and record the project directory, or open an existing Unity project.

1.2 Open the project with an IDE (such as Visual Studio Code):
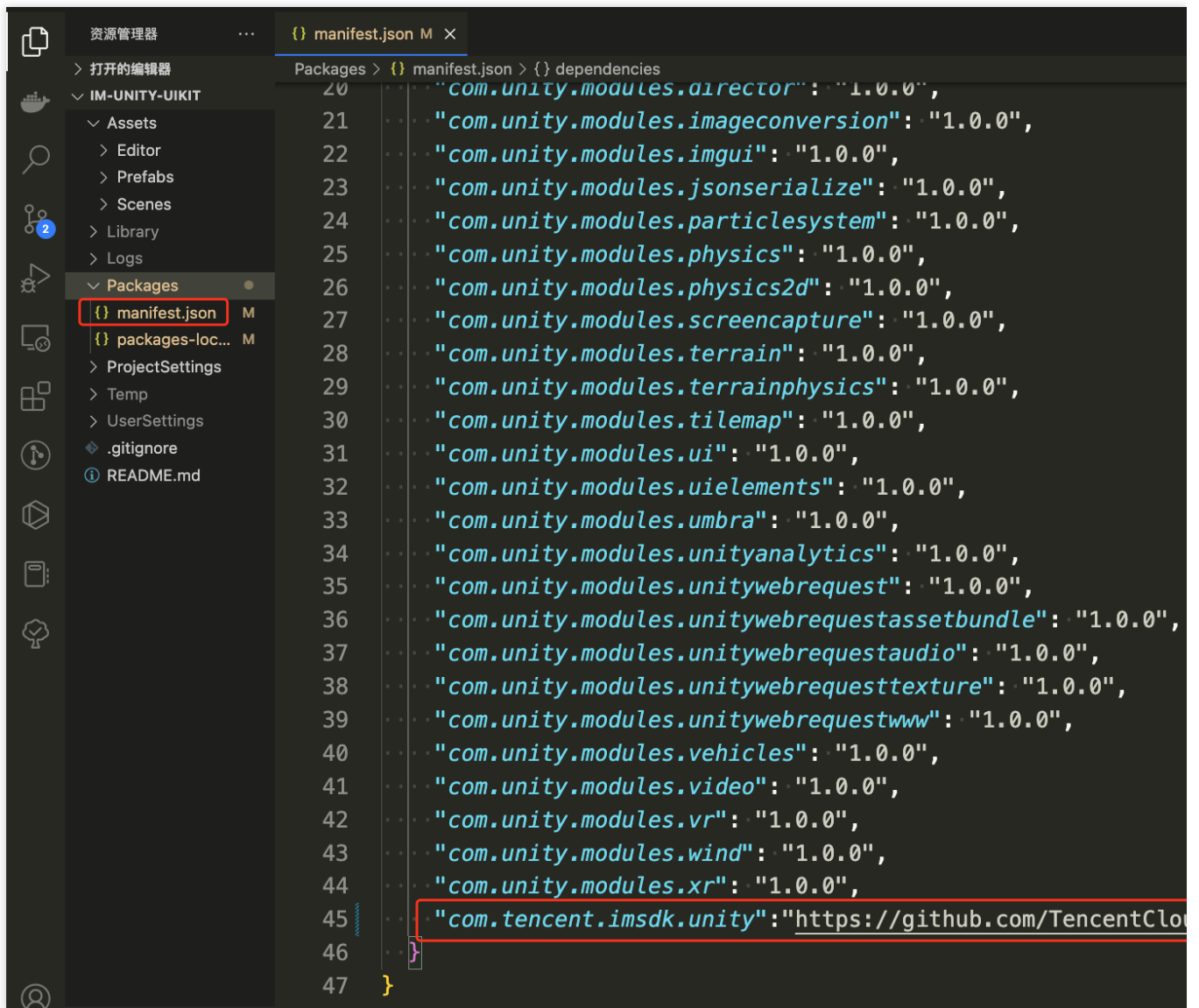
1.3 Find Packages/manifest.json based on the directory, and modify dependencies as follows:
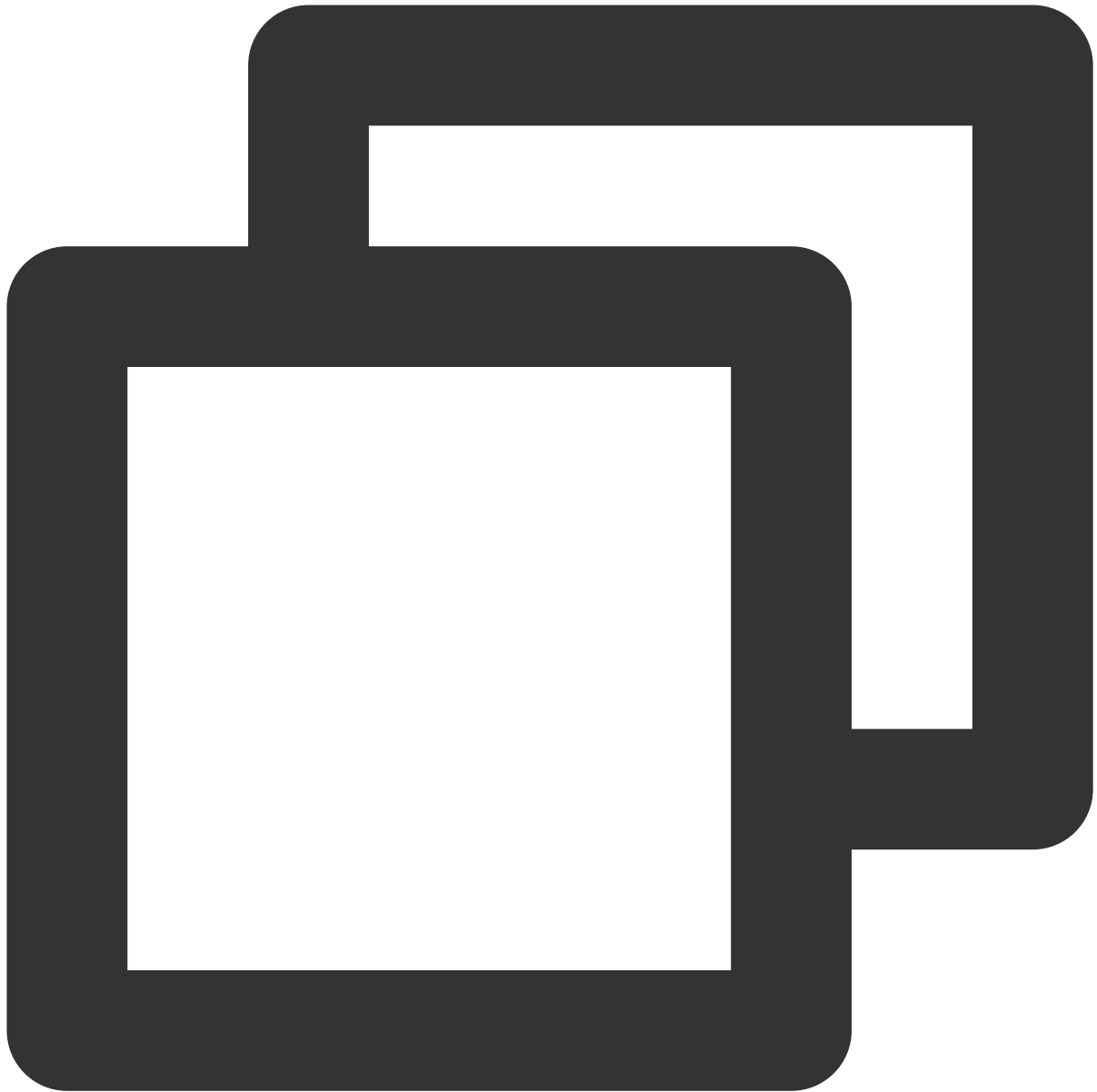
```
{
  "dependencies":{
    "com.tencent.imsdk.unity":"https://github.com/TencentCloud/chat-sdk-unity.git#u
  }
}
```

To help you better understand Chat SDK APIs, we provide API examples to demonstrate how to call APIs and trigger listeners.

Part 3. Loading Dependencies

Open the project in the Unity Editor, wait until dependencies are loaded, and confirm the Tencent Cloud Chat is successfully loaded.
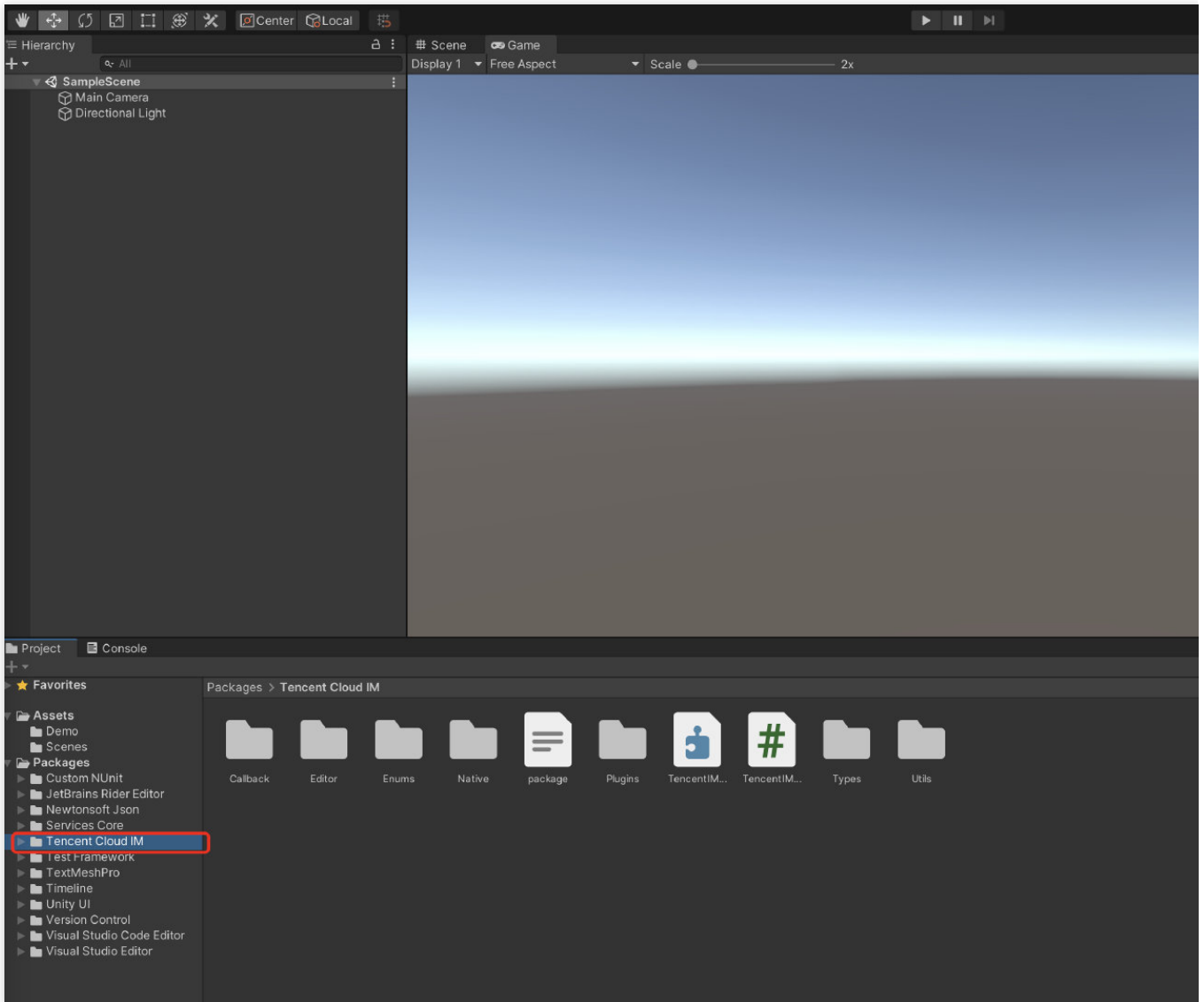
Part 4. Implementing Your Own UI

Prerequisites

You have created a Unity project or have a project that can be based on Unity, and have loaded Tencent Cloud Chat SDK.

Initializing the SDK

[Detailed documentation](#)

Call `TencentIMSDK.Init` to initialize the SDK.

Pass in your `SDKAppID` .

```
using Com.Tencent.IM.Unity.UIKit;
using com.tencent.imsdk.unity;
using com.tencent.imsdk.unity.types;
using com.tencent.imsdk.unity.enums;
namespace Com.Tencent.IM.Unity.UIKit{
```

```
    public static void Init() {
        string SDKAppID = ""; // Get the `SDKAppID` from the Chat console
        SdkConfig sdkConfig = new SdkConfig();

        sdkConfig.sdk_config_config_file_path = Application.persistentDataPath + "/TI

        sdkConfig.sdk_config_log_file_path = Application.persistentDataPath + "/TIM-L

        TIMResult res = TencentIMSDK.Init(long.Parse(SDKAppID), sdkConfig);
    }
}
```

After `Init` , you can mount some listeners to the Chat SDK, mainly including those for network status and user information change. For more information, see here.

Logging in with a test account

Detailed documentation

Log in with one of the test accounts you created earlier.

Call the `TencentIMSDK.Login` method to log in with the test account.

If the returned `res.code` is `0` , the login is successful.
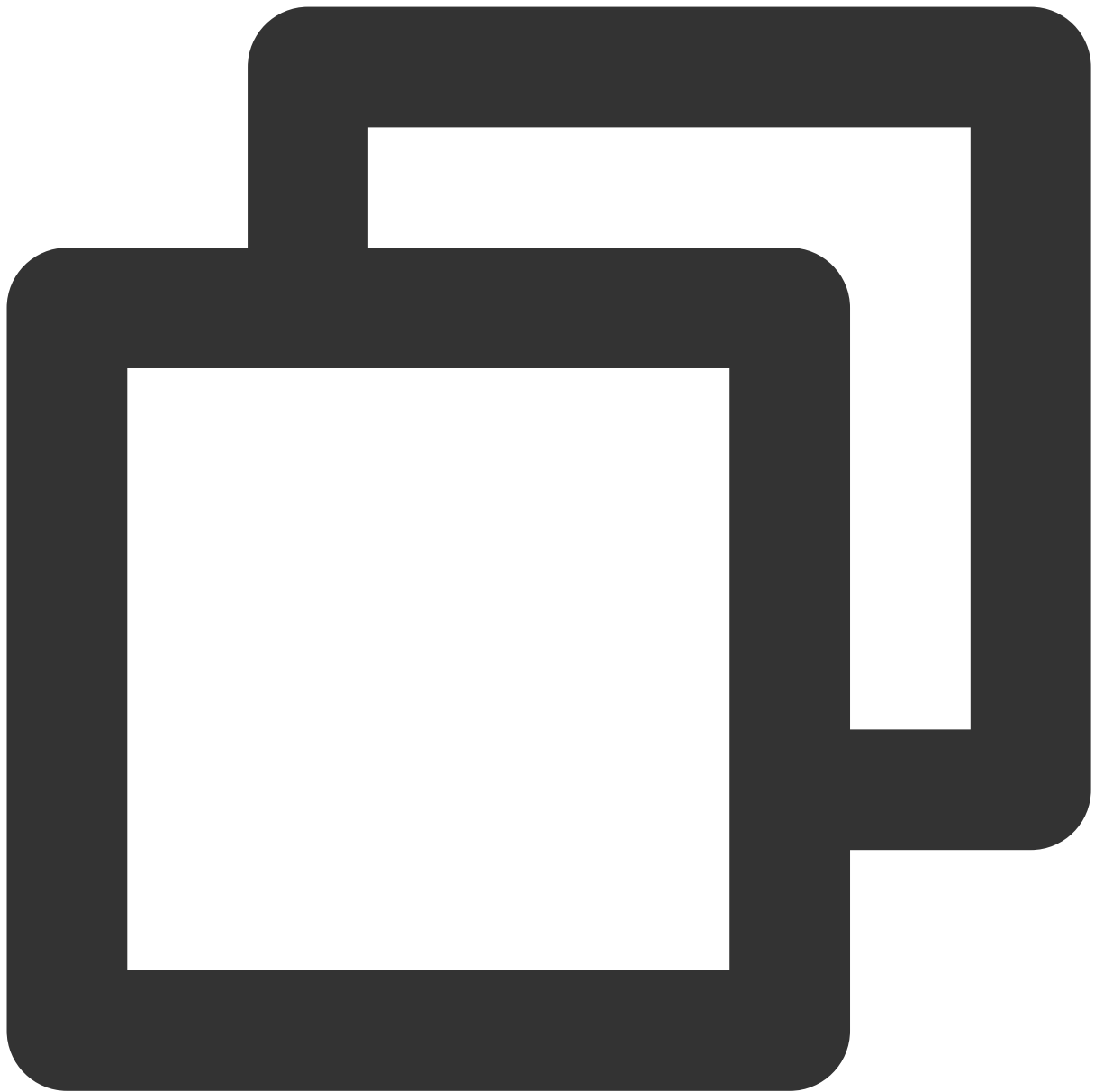
```
public static void Login() {
    if (userid == "" || user_sig == "")
    {
        return;
    }
    TIMResult res = TencentIMSDK.Login(userid, user_sig, (int code, string desc, stri
        // Process the login callback logic
    });
}
```

Note

The test account is for development and testing only. Before the application is launched, the correct method for generating a UserSig is to integrate the UserSig calculation code into your server and provide an application-oriented API. When UserSig is needed, your application can send a request to the business server for a dynamic UserSig. For more information, see Generating UserSig.

Sending a message

Detailed documentation

The following shows how to send a text message:

Sample code:
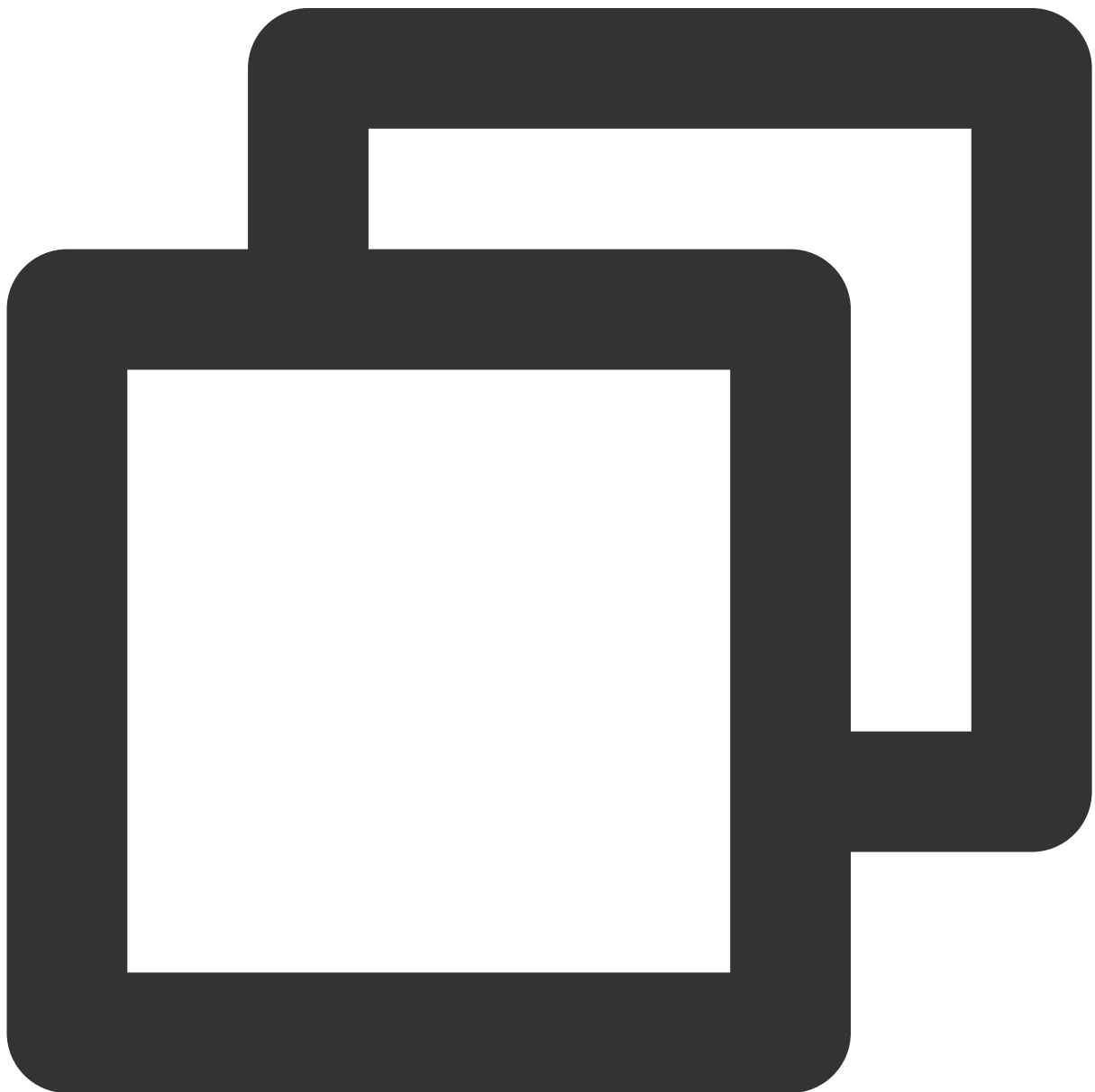
```
public static void MsgSendMessage() {
        string conv_id = ""; // The conversation ID of a one-to-one message is the
        Message message = new Message
        {
          message_conv_id = conv_id,
          message_conv_type = TIMConvType.kTIMConv_C2C, // For a group message, thi
          message_elem_array = new List<Elem>
          {
            new Elem
            {
              elem_type = TIMElemType.kTIMElem_Text,
              text_elem_content =  "This is an ordinary text message"
            }
          }
        };
        StringBuilder messageId = new StringBuilder(128); // messageId for getting

        TIMResult res = TencentIMSDK.MsgSendMessage(conv_id, TIMConvType.kTIMConv_C
          // Async message sending result
        });
          // The message ID returned when the message is sent
}
```

Note

If sending fails, it may be that your `sdkAppID` doesn't support sending messages to strangers. In this case, you can disable the relationship check feature in the console.

Disable the friend relationship chain check here.

Obtaining the conversation list

Detailed documentation

Log in with the second test account to pull the conversation list.

The conversation list can be obtained in two ways:

1.1 Listen for the persistent connection callback to get message changes and update and render the historical message list in real time.

1.2 Call an API to get the message history at certain time points.

Common use cases include:

Getting the conversation list when the application starts and listening for the persistent connection callback to update the conversation list in real time.

Requesting the conversation list at certain time points
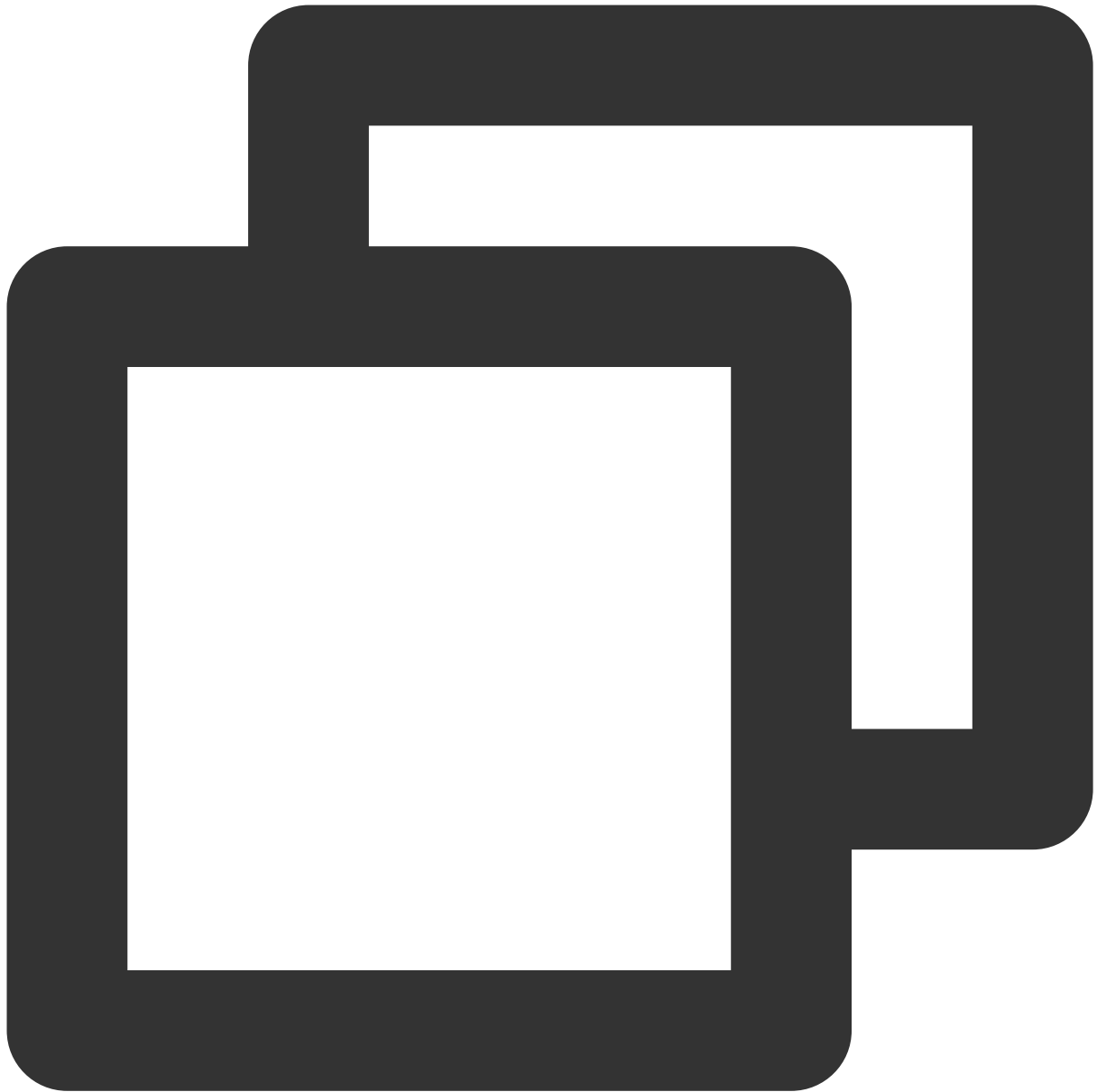
```
TIMResult res = TencentIMSDK.ConvGetConvList((int code, string desc, List<ConvInfo>
 // Process the async logic
});
```

At this point, you can see the message sent by the first test account in the previous step.

Listening for the persistent connection to get the conversation list in real time

Mount the conversation list listener, process the callback event, and update the UI.

Mount the listener.
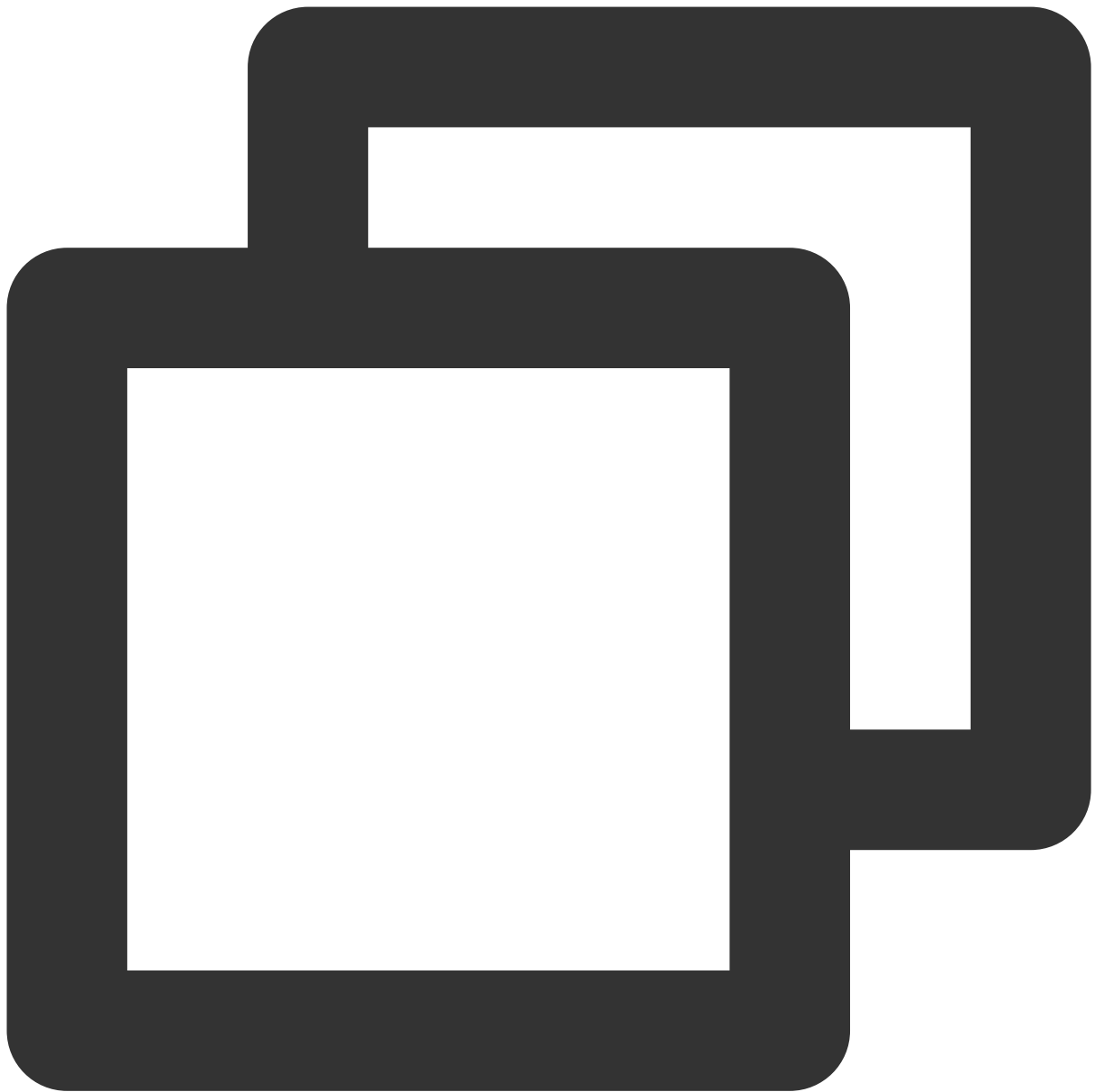
```
TencentIMSDK.SetConvEventCallback((TIMConvEvent conv_event, List<ConvInfo> conv_lis
  // Process the callback logic
});
```

Process the callback event and display the latest conversation list on the UI.

Receiving messages

[Detailed documentation](#)

Messages can be received with the Chat SDK in two ways:

Listen for the persistent connection callback to get message changes and update and render the historical message list in real time.

Call an API to get the message history at certain time points.

Common use cases include:

After a new conversation is opened on the UI, request and display a certain number of historical messages.

Listen for the persistent connection to receive messages in real time and add them to the historical message list.

Requesting the historical message list at a time

To avoid affecting the pull speed, we recommend you limit the number of messages pulled to 20 per page.

You need to dynamically record the current number of pages for the next request.

Sample code:
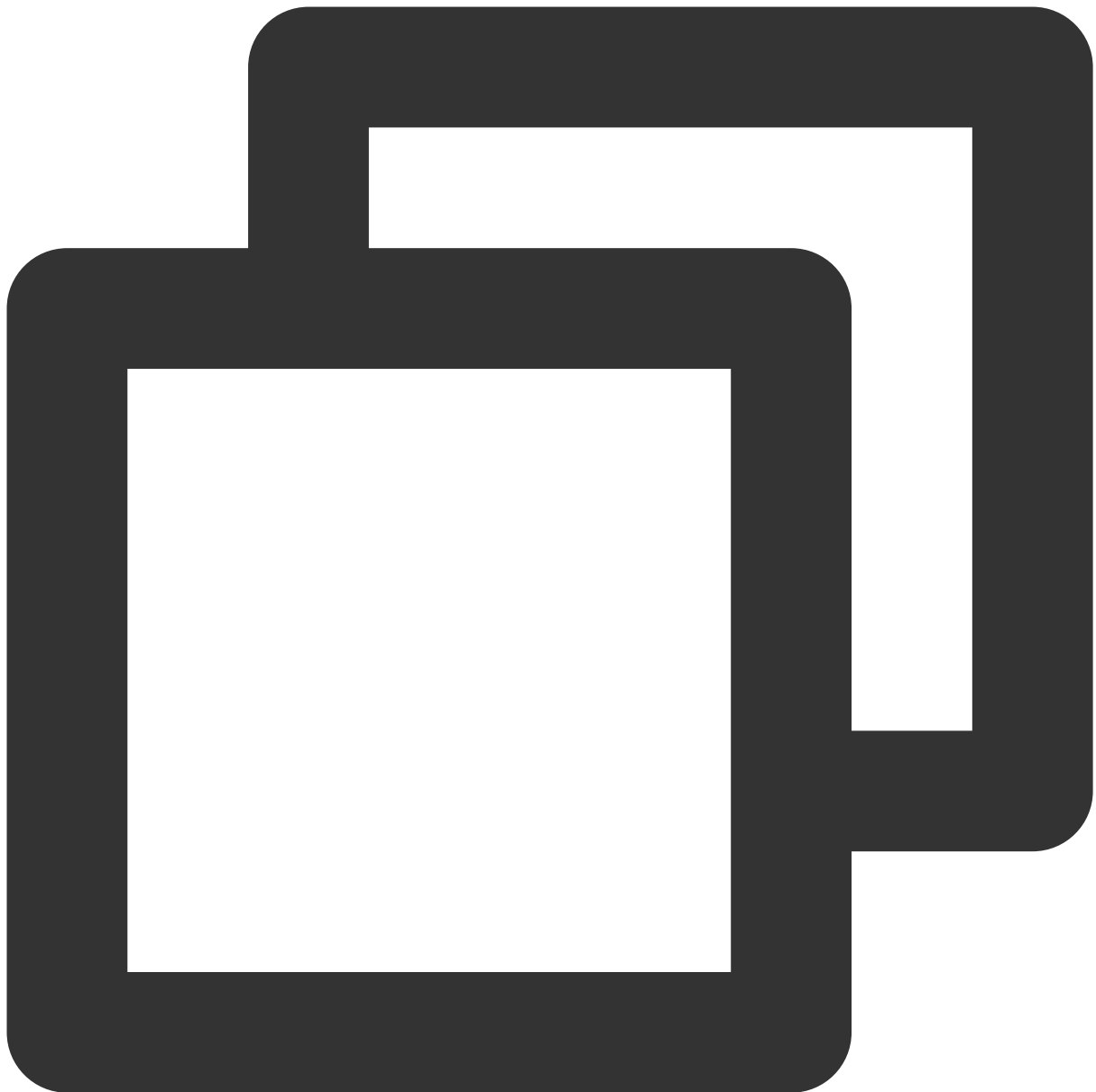
```
// Pull historical one-to-one messages
// Do not set `msg_getmsglist_param_last_msg` for the first pull. It will pull the
// `msg_getmsglist_param_last_msg` can be the last message in the returned message
Message LastMessage = null;
string LastMessageID = "";
var get_message_list_param = new MsgGetMsgListParam();
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_C2C, get_m
  // handle callback logic
  List<Message> messages = Utils.FromJson<List<Message>>((string)parameters[1]);
   if (messages.Count > 0){
     LastMessage = messages[messages.Count - 1];
```

```
      LastMessageID.text = messages[messages.Count - 1].message_msg_id;
    }else {
      LastMessage = null;
      LastMessageID.text = "";
    }

});
// `msg_getmsglist_param_last_msg` can be the last message in the returned message
var get_message_list_param = new MsgGetMsgListParam
    {
      msg_getmsglist_param_last_msg = LastMessage
    };
TIMResult res = TencentIMSDK.MsgGetMsgList(conv_id, TIMConvType.kTIMConv_Group, get
  // handle callback logic
});
```

Listening for the persistent connection callback to get new messages in real time

After the historical message list is initialized, new messages are from the persistent connection

`TencentIMSDK.AddRecvNewMsgCallback` .

After the `AddRecvNewMsgCallback` callback is triggered, you can add new messages to the historical message list as needed.

Sample code for binding a listener:

Tencent Cloud


```
TencentIMSDK.AddRecvNewMsgCallback((List<Message> message, string user_data) => {
  // Process new messages
});
```

At this point, you have completed the Chat module development, and now users can send and receive messages and enter different conversations.

You can develop more features, such as group, user profile, relationship chain, and local search.

For more information, see Integration Solution (No UI).

Part 5. Enabling Unity Support for WebGL

Tencent Cloud Chat SDK for Unity 1.8.1 or later supports building WebGL.

To enable support for web, you need to perform the following extra steps in addition to those for enabling support for Android and iOS:

Importing JS

Download the following three JS files from Npm and place them in the WebGL artifacts directory under the project:

tim-js

tim-js-friendship.js

Rename this file to tim-upload-plugin.js

Open `index.html` and import the three JS files as follows:

```
<script src="./tim-js.js"></script>
<script src="./tim-js-friendship.js"></script>
<script src="./tim-upload-plugin.js"></script>
```
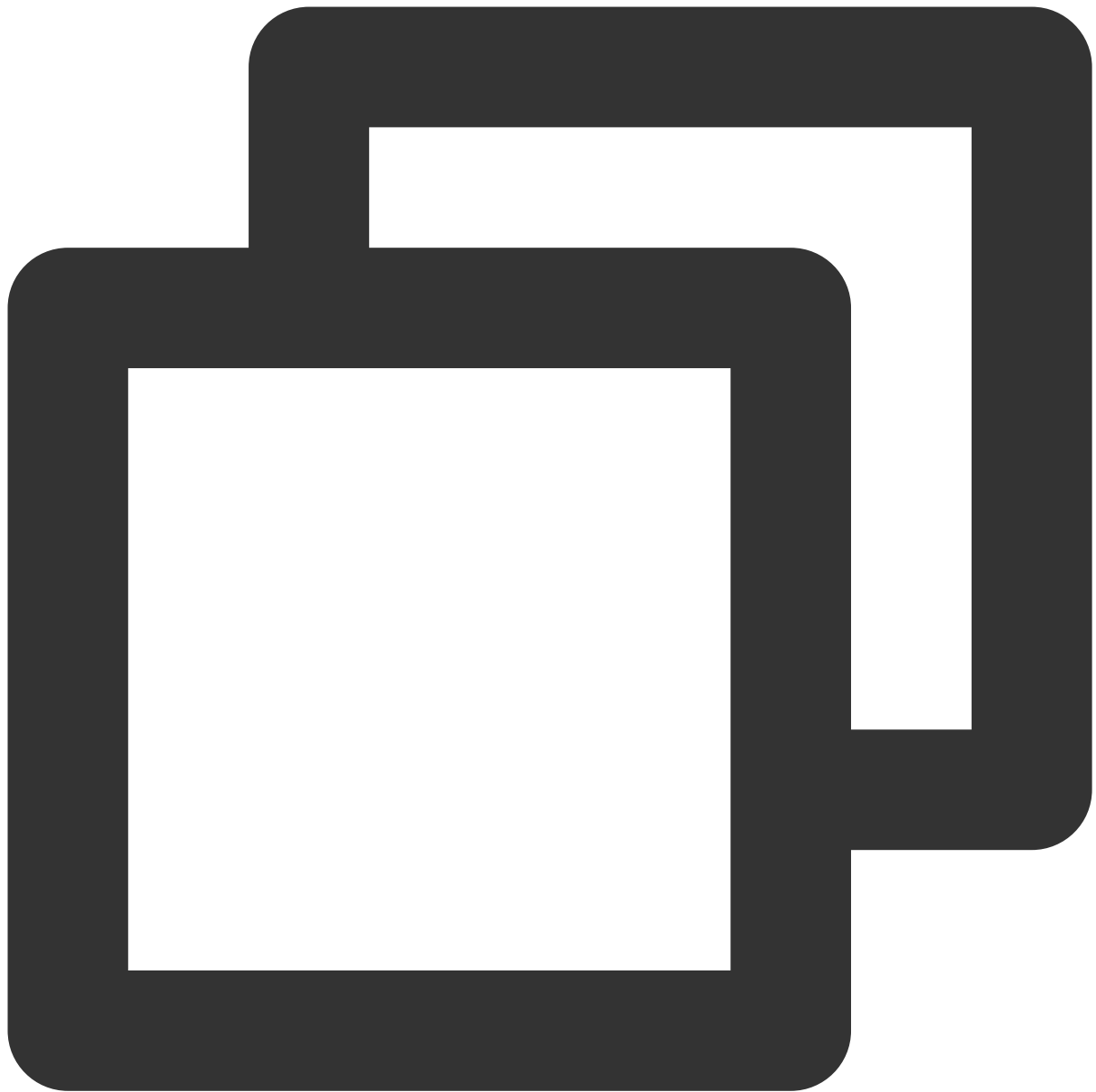
FAQs

What platforms are supported?

iOS, Android, Windows, macOS, and WebGL are supported.

What should I do if clicking Build And Run for an Android device triggers an error, stating no available device is found?

Check that the device is not occupied by other resources. Alternatively, click **Build** to generate an APK package, drag it to the simulator, and run it.

What should I do if an error occurs during the first run for an iOS device?

If an error is reported for an iOS device after the demo configured as above is run, choose **Product** > **Clean** to clean the product, and build the demo again. You can also close Xcode and open it again, and then build the demo again.

What should I do if Unity v2019.04 reports the following error for iOS?

Library/PackageCache/com.unity.collab-proxy@1.3.9/Editor/UserInterface/Bootstrap.cs(23,20): error CS0117: 'Collab' does not contain a definition for 'ShowChangesWindow'
Choose **Window** > **Package Manager** on the toolbar of the Editor and downgrade Unity Collaborate to 1.2.16.

What should I do if Unity v2019.04 reports the following error for iOS?

Library/PackageCache/com.unity.textmeshpro@3.0.1/Scripts/Editor/TMP_PackageUtilities.cs(453,84): error CS0103: The name 'VersionControlSettings' does not exist in the current context
Open the source code and delete the code snippet of `|| VersionControlSettings.mode != "Visible Meta Files"` .

Is this a C# interface? How to use it without Unity?

Unity SDK is an SDK using C#, but because Unity SDK contains Unity features, it cannot be used directly in a C# environment.
If you need to use it in a C# environment, we provide a separate C# SDK nuget package. The usage method is the same as Unity SDK, you can refer to Unity SDK documentation for use.
Among them, the C# SDK only supports the PC side, and the unity SDK supports the mobile side.

Is there a UI that can be used directly?

The corresponding UIKit of untiy SDK and C# SDK is currently not provided.

How do I query error codes?

For Chat SDK API error codes, see Error Codes.

UE

Last updated：2024-04-10 17:47:55

This document describes how to quickly run the Chat demo for Unreal Engine.

Note:

 Currently, the demo can be run on Windows, macOS, iOS, and Android.

Environment Requirements

Unreal Engine 4.27.1 or later.

Platform	Environment
Android	Android Studio 4.0 or later. Visual Studio 2017 15.6 or later. A real device for testing.
iOS & macOS	Xcode 11.0 or later. OSX 10.11 or later. Ensure your project has a valid developer signature.
Windows	OS: Windows 7 SP1 or later (64-bit based on x86-64). Disk capacity: At least 1.64 GB free space after the IDE and required tools are installed. Install Visual Studio 2019.

Prerequisites

You have signed up for a Tencent Cloud account and completed identity verification.

Directions
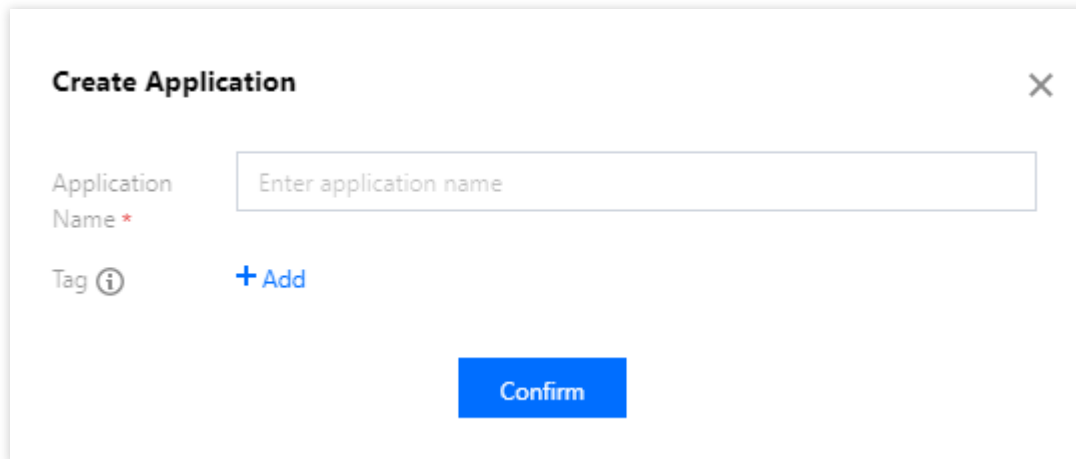
Step 1. Create an app

1. Log in to the Chat console.

Note:

If you already have an app, record its SDKAppID and obtain key information.

A Tencent Cloud account can create a maximum of 300 Chat apps, after which you need to disable and delete an unwanted app before creating a new one. **Once an app (along with its SDKAppID) is deleted, the service it provides and all its data are lost. Proceed with caution**.

2. Click **Create Application**, enter your app name, and click **Confirm**.

3. After creation, you can see the status, service version, SDKAppID, creation time, tag, and expiry time of the new app on the overview page of the console. Record the SDKAppID.

Step 2. Obtain key information

1. Click the app to go to its basic configuration page.

2. In the **Basic Information** area, click **Display key**, and then copy and save the key information.

Caution:

Make sure you keep your key information properly secure to prevent disclosure.

Step 3. Configure the demo project file

1. Download the Chat demo project from Download the Demo.

2. Find and open `/IM_Demo/Source/debug/include/DebugDefs.h` .

3. Set parameters in `DebugDefs.h` as follows:

Note:

In this document, the method to obtain UserSig is to configure a SECRETKEY in the client code. In this method, the SECRETKEY is vulnerable to decompilation and reverse engineering. Once your SECRETKEY is disclosed, attackers can steal your Tencent Cloud traffic. Therefore, **this method is only suitable for locally running a demo project and feature debugging**.

The correct `UserSig` distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see Generating UserSig on the Server.

Step 4. Compile, package, and run the project

1. Double-click to open `/IM_Demo/IM_Demo.uproject`.

2. Compile, run, and test the project.

macOS

Windows

iOS

Android

File -> **Package Project** -> **Mac**

File -> **Package Project** -> **Windows** -> **Windows(64-bit)**
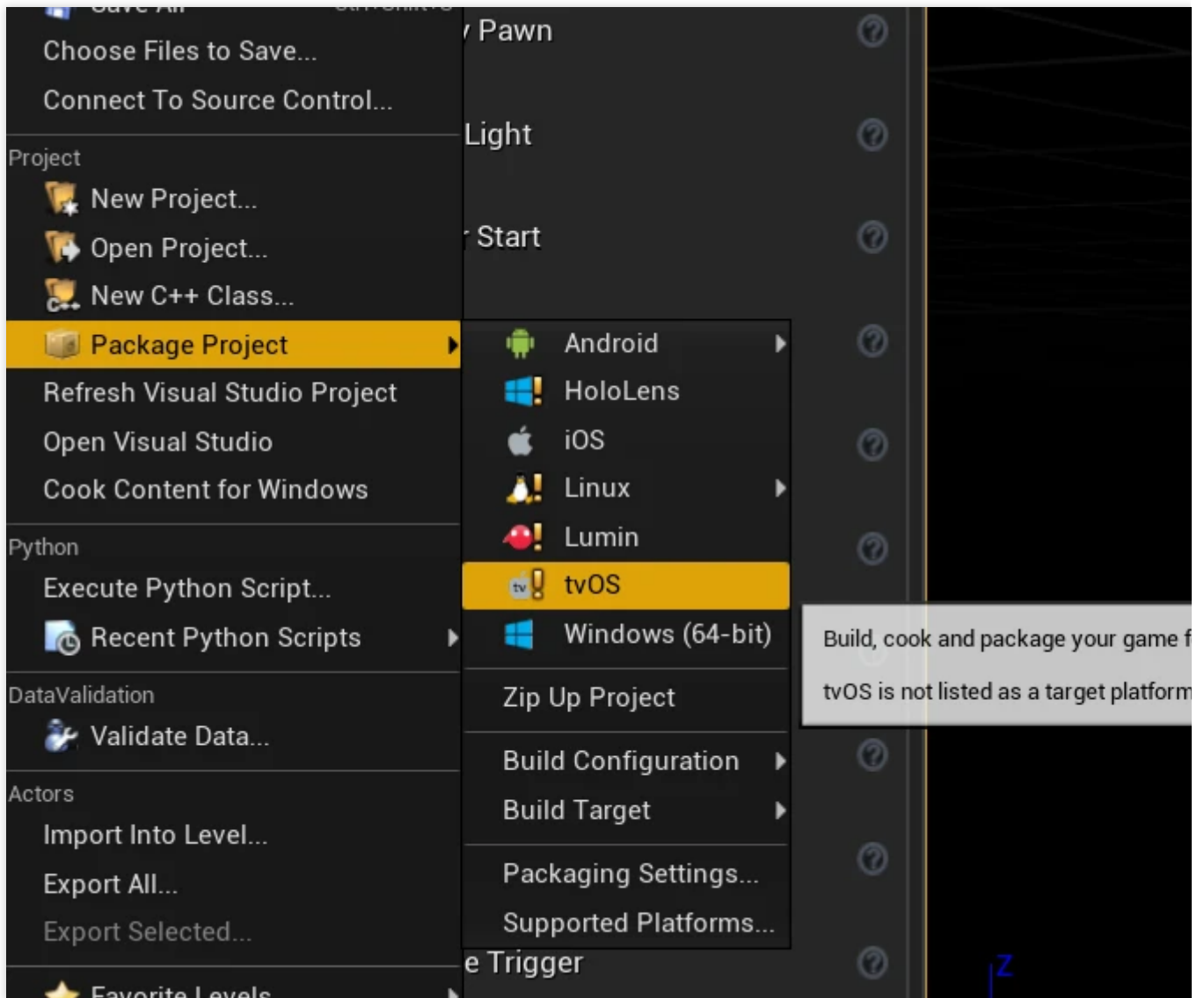
Package Project

File -> **Package Project** -> **Mac**

1. For development and testing, see Android Quick Start.

2. For packaging, see Packaging Android Projects.

Chat API Documentation for Unreal Engine

For more information on APIs, see API Overview.

FAQs

Tencent Cloud

What should I do if the error "Attempt to construct staged filesystem reference from absolute path" occurs on Android?

Close the project in UE4, open CMD, and run the following commands:
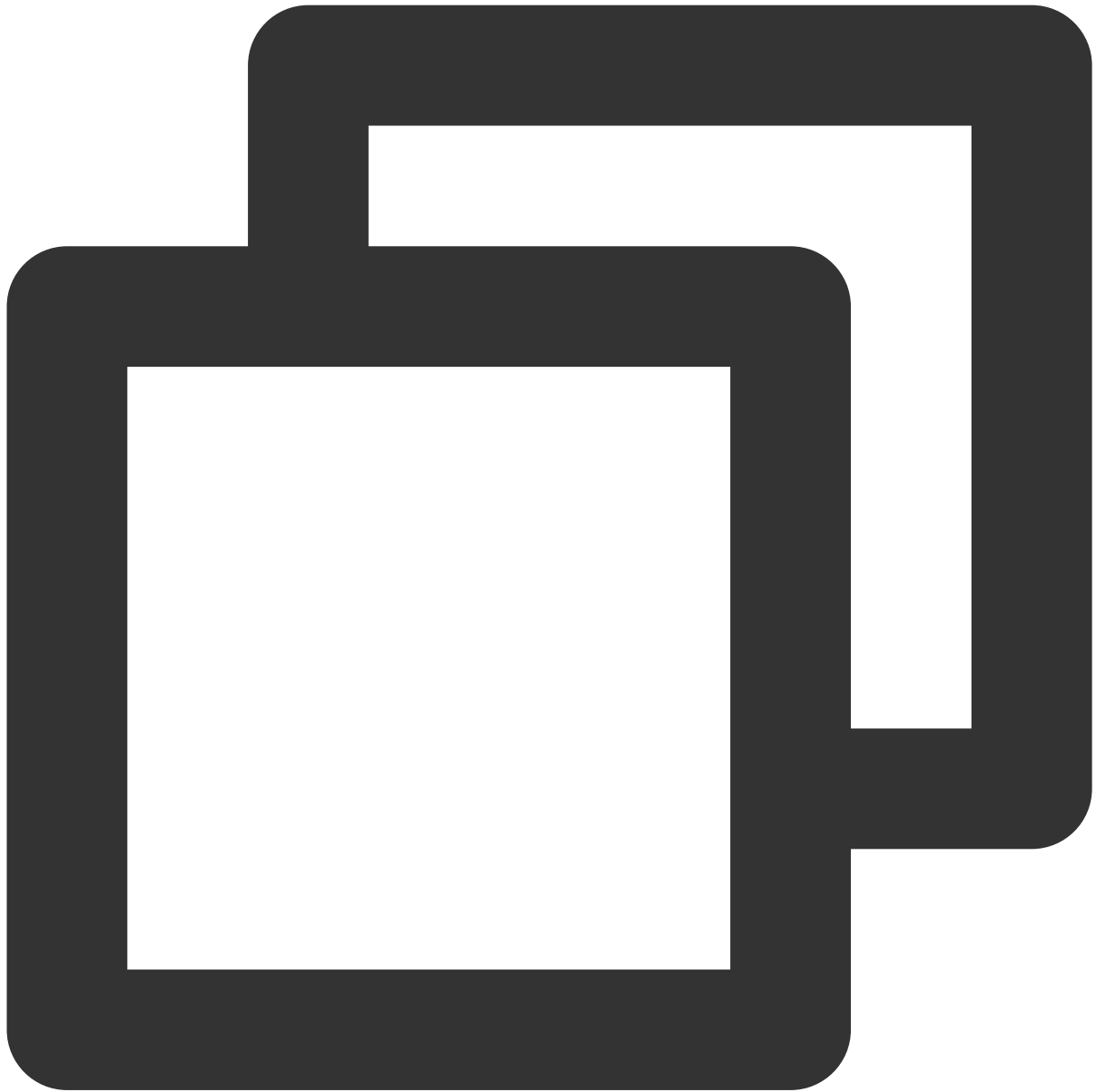

```
adb shell

cd sdcard

ls (you should see the UE4Game directory listed)

rm -r UE4Game
```

Compile your project again.

Flutter

Last updated：2024-04-23 18:34:18

This document describes how to integrate the SDK for Flutter.

Environment Requirements

Environment	Version
Flutter	Flutter 3.0.0 or later for the Chat SDK; Flutter 3.16.0 or later for the TUIKit component library.
Android	Android Studio Dolphin | 2021.3.1 or later; and devices with Android 7.0 or later for apps
iOS	Xcode 12.0 or later. Ensure that your project has a valid developer signature.

Supported Platforms

We offer a set of Chat SDK and UIKit for all Flutter platforms, allowing you to run one set of code on all platforms.

Platform	Low-level SDK (tencent_cloud_chat_sdk)	UIKit (tencent_cloud_chat_uikit) |	UIKit V2 (tencent_cloud_chat)
iOS	Supported	Supported	Supported
Android	Supported	Supported	Supported
Web	Supported from v4.1.1+2	Supported from v0.1.5	*Coming Soon...*
macOS	Supported from v4.1.9	Supported from v2.0.0	Supported
Windows	Supported from v4.1.9	Supported from v2.0.0	Supported
Hybrid development (Add the SDK for Flutter to your existing native app)	Supported from v5.0.0	Supported from v1.0.0	Supported

Note:

For Web and macOS integration requires a few extra steps. For more information, see Expanding to More Platforms.

Prerequisites

1. You have signed up for a Tencent Cloud account and completed identity verification.

2. You have created an application as instructed in Creating and Upgrading an Application and recorded the `SDKAppID` .

3. In the Chat console, select your application and click **Auxiliary Tools** > **UserSig Generation & Verification** on the left sidebar. Create two `UserID` values and their `UserSig` values, and copy the `UserID` , `Key` , and `UserSig` so they can be used to log in later.

Note:

This account is for development and testing only. Before the application is launched, the correct `UserSig` distribution method is to use the server to generate `UserSig` and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see Generating UserSig.

Selecting a Suitable Scheme to Integrate the SDK for Flutter

Chat

offers three integration schemes:

Integration Scheme	Applicable Scenario
Modifying the sample app	The Chat sample app is a complete chat application built with UIKit. If you need to implement chat scenarios, you can use the sample app for secondary development. Try it out here.
Using UI library	The Chat UI component library `UIKit` provides common UI components, such as conversation list, chat UI, and contacts. You can use the component library to quickly build a custom Chat application as needed. **This scheme is recommended**.
Implementing UI on your own	You can use the Low-level SDK, if UIKit cannot meet your UI requirements or you want to further customize your UI.

Scheme 1. Modifying the Sample App

This sample app is built with the UIKit V2.

Running Sample App

Tencent Cloud

1. Download the source code and install dependencies:
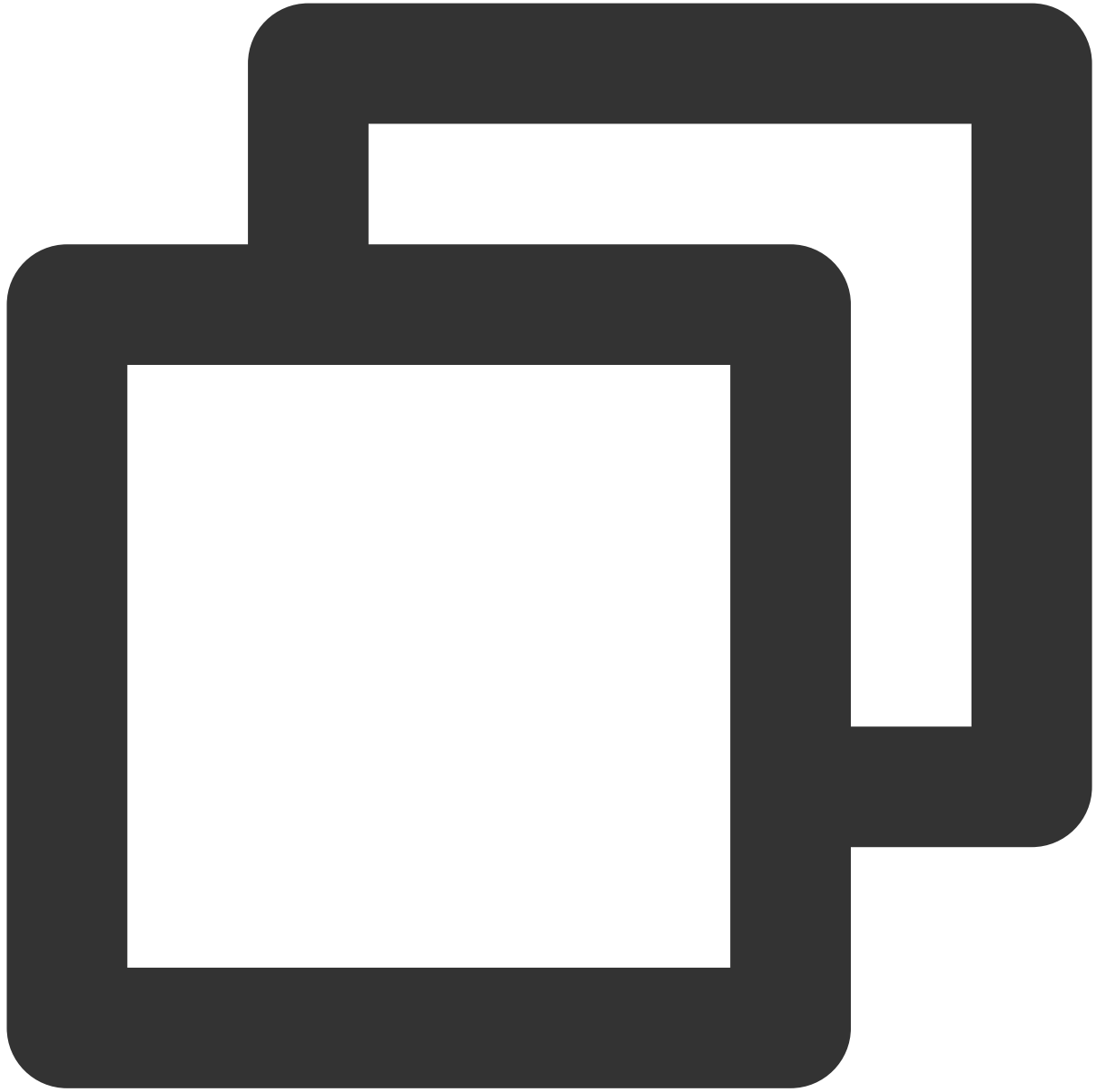

```
# Clone the code
git clone https://github.com/TencentCloud/chat-demo-flutter.git

# Checkout the 'v2' branch
git checkout v2

# Clean the project. Important
flutter clean
```

```
# Install dependencies
flutter pub get
```

2. Configure the user info for login.

Open `lib/config.dart` , and specify the `sdkappid` , `userid` , and `usersig` obtained and generated in the previous step.

3. Run the app.
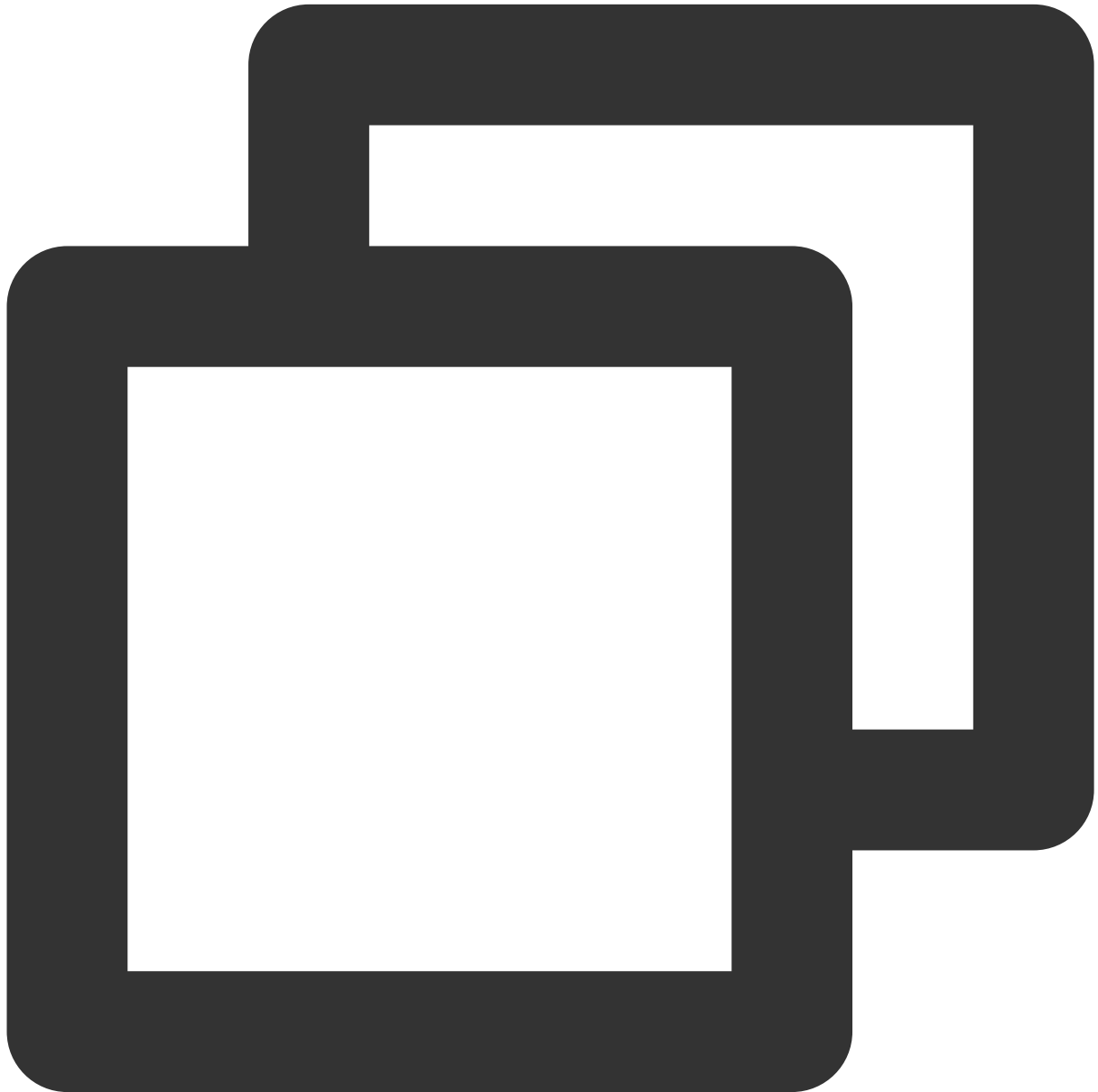

```
flutter run
```

Sample app code structure

Folder	Features
lib	Program core directory
lib/desktop	Desktop adaptation related code
lib/setting	The related code for Profile and Settings pages

The `lib` directory's `lib/main.dart` file is the core file of this sample app program. As you can see, there are several lines of key code. The `List<Widget> pages` component list is passed into `bottomNavigation` for page switching.
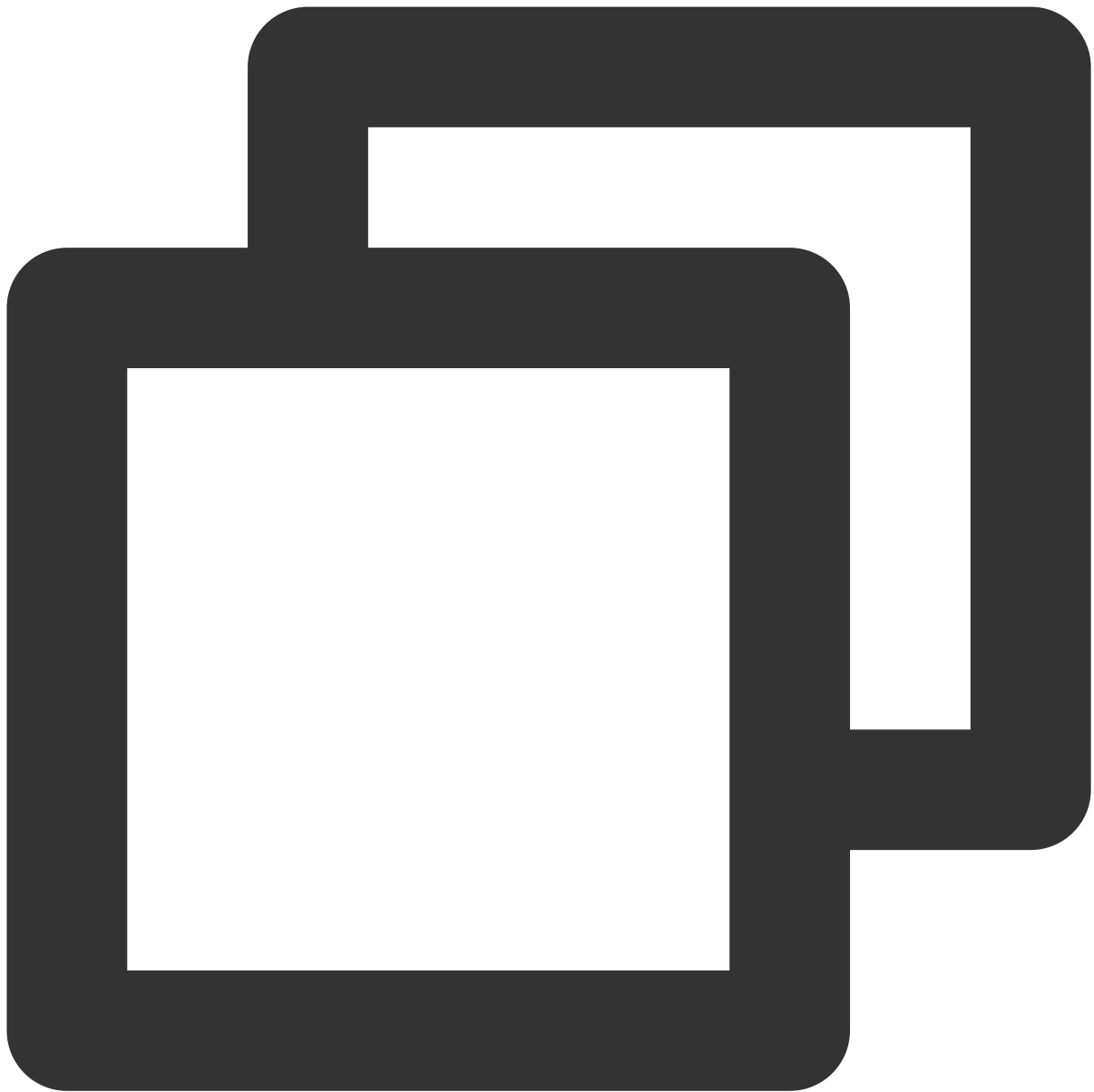
```
pages = [
  const TencentCloudChatConversation(),
  const TencentCloudChatContact(),
  TencentCloudChatSettings(
    removeSettings: removeLocalSetting,
    setLoginState: changeLoginState,
  ),
];
```

Although the new version of UIKit has many components, these components are built-in support for routing and navigation. That is, to minimize the use, only manually use the `TencentCloudChatConversation` conversation list component and `TencentCloudChatContact` contacts component, you can use all associated UIKit components, including the Message, Contact, and Group Profile components, etc. These components can be navigated to through the components you manually import.

If you have more advanced customization requirements for using the UIKit, you can refer to the quick start document of UIKit.

Scheme 2. Using UIKit Component Library to Quickly Integrate Chat Capabilities

Our new Flutter Chat UIKit V2 provides developers with a comprehensive set of tools to easily create feature-rich chat applications.

It is built with a modular approach, allowing you to pick and choose the components you need while keeping your application lightweight and efficient.

The UIKit includes a wide range of capabilities, such as Conversation List, Message handling, Contact lists, User and Group Profiles, Search functionality, and more.
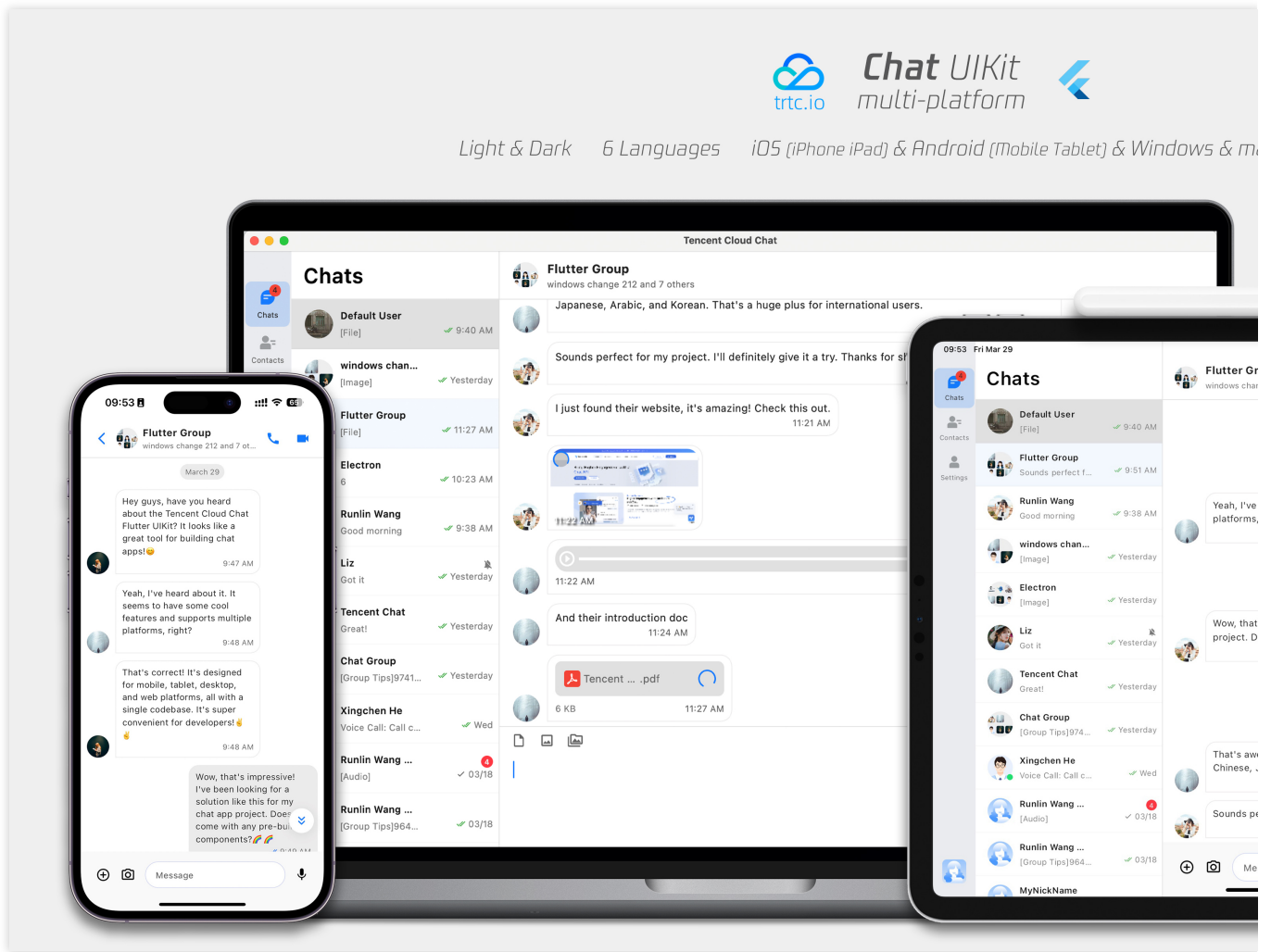
This section provides a brief overview of how to integrate and use the UIKit. For a more comprehensive guide, please refer to the Quick Start documentation provided Integrating Basic Features - Flutter.

Compatibility

Our UIKit supports both **mobile**, **tablet** and **desktop** UI styles, and is compatible with Android, iOS, macOS, Windows, and Web (will be supported in future versions).
It comes with built-in support for English, Simplified Chinese, Traditional Chinese, Japanese, Korean, and Arabic languages (with support for Arabic RTL interface), and light and dark appearance styles.

Requirements

Flutter version: 3.16 or above

Dart version: 3.0 or above

Getting Started

To start using our UIKit, first import the base package, tencent_cloud_chat.
Next, import the required UI component packages that suit your needs from the following list:

[tencent_cloud_chat_message](#)

[tencent_cloud_chat_conversation](#)

[tencent_cloud_chat_contact](#)

[tencent_cloud_chat_user_profile](#)

[tencent_cloud_chat_group_profile](#)

tencent_cloud_chat_search *(In Beta)*

The architecture of our UIKit is shown below:
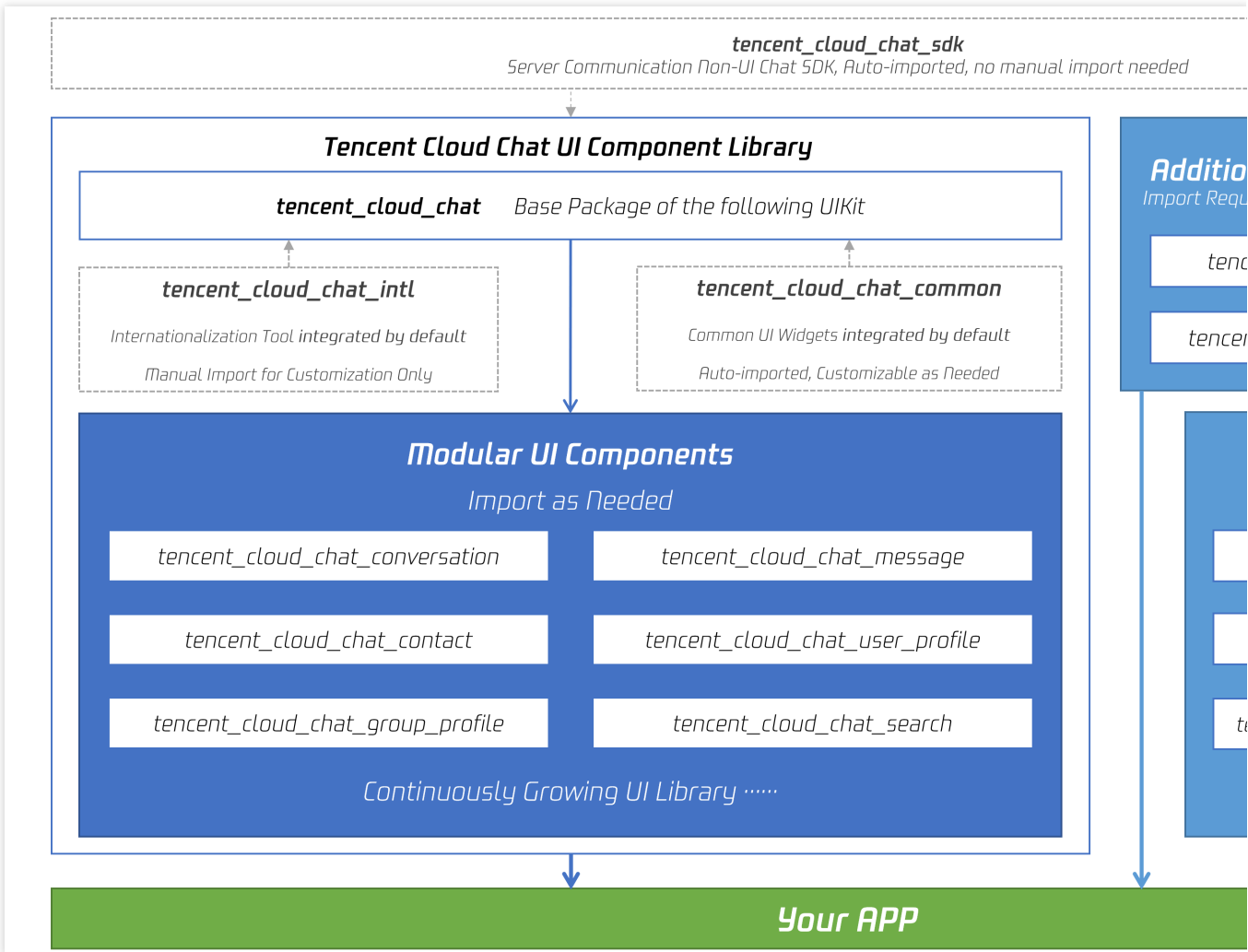
Integration

For the integration of Chat UIKit, see [Integrating Basic Features - Flutter](#).

Scheme 3. Implementing Your Own UI

Prerequisites

You

 have created a Flutter application or have an existing application that can be based on Flutter.

Directions

Installing the Chat SDK

In this scheme, the term "Chat SDK" refers to the Low-level SDK.

Detailed documentation

Run the following command to install the latest version of the Chat SDK for Flutter.
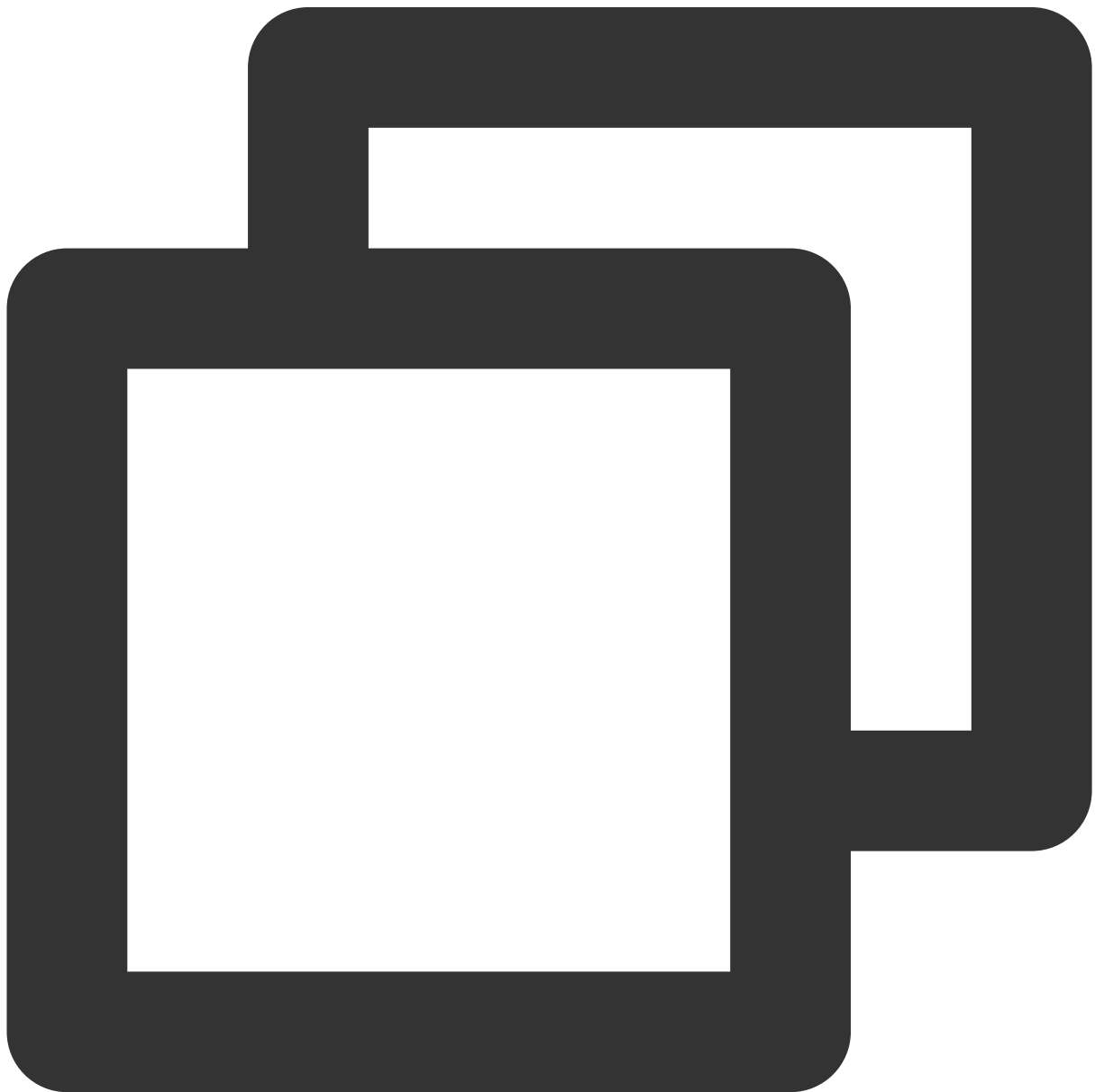
```
flutter pub add tencent_cloud_chat_sdk
```

Note:

If your project also needs to be applied to web or desktop (macOS/Windows) platforms, you need to perform a few extra steps for SDK integration.

Initializing the SDK

Detailed documentation

Call `initSDK` to initialize the SDK.

Pass in your `sdkAppID` .
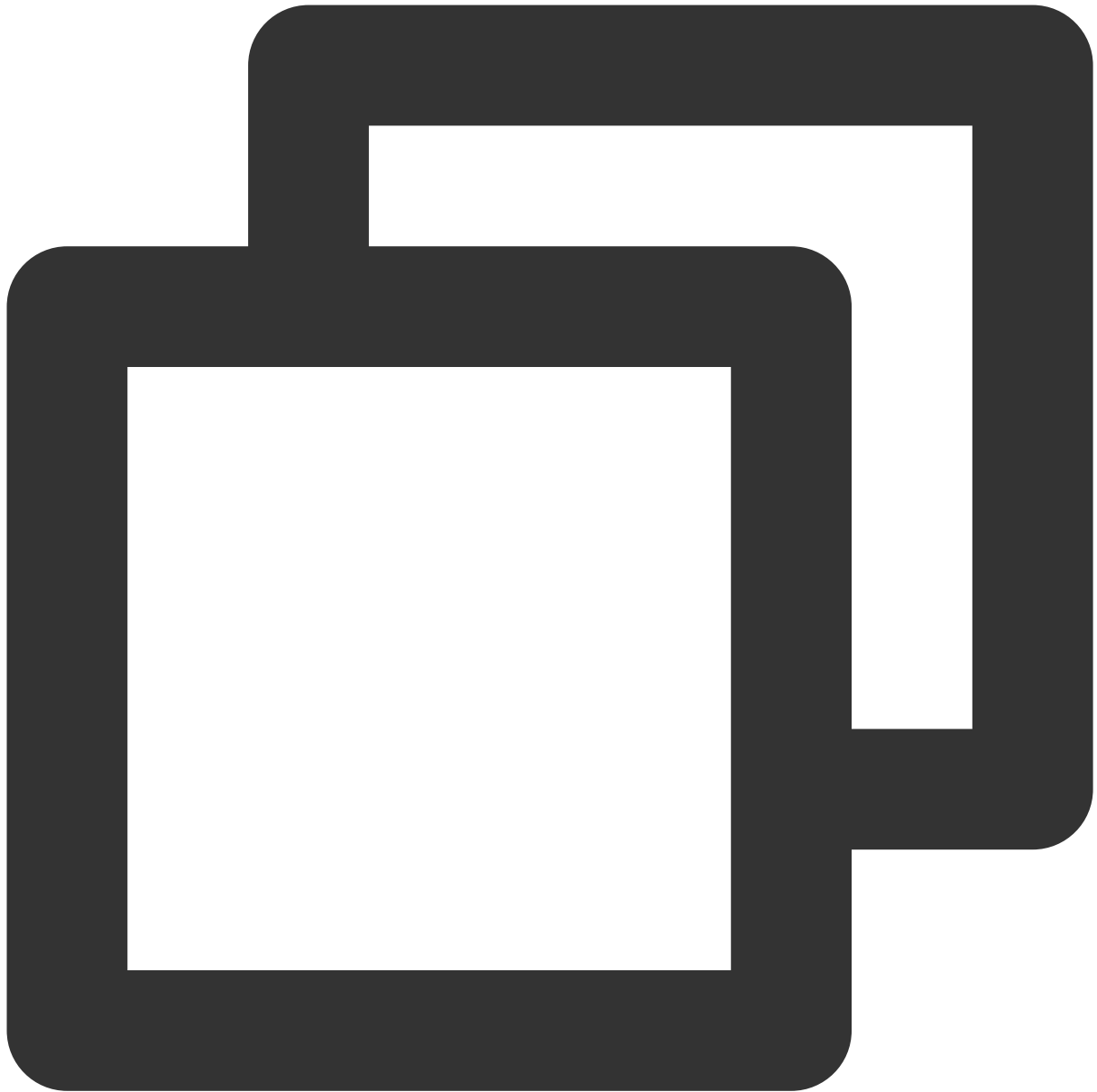
```
import 'package:tencent_cloud_chat_sdk/enum/V2TimSDKListener.dart';
import 'package:tencent_cloud_chat_sdk/enum/log_level_enum.dart';
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';
TencentImSDKPlugin.v2TIMManager.initSDK(
  sdkAppID: 0, // Replace 0 with the SDKAppID of your Chat application when integra
  loglevel: LogLevelEnum.V2TIM_LOG_DEBUG, // Log
  listener: V2TimSDKListener(),
);
```

In this step, you can mount some listeners to the Chat SDK, mainly including those for network status and user information changes. For more information, see here.

Logging in with a test account

Detailed documentation

Log in with a test account initially generated for login verification.

Call the `TencentImSDKPlugin.v2TIMManager.login` method to log in with the test account.

If the returned `res.code` is `0`, the login is successful.
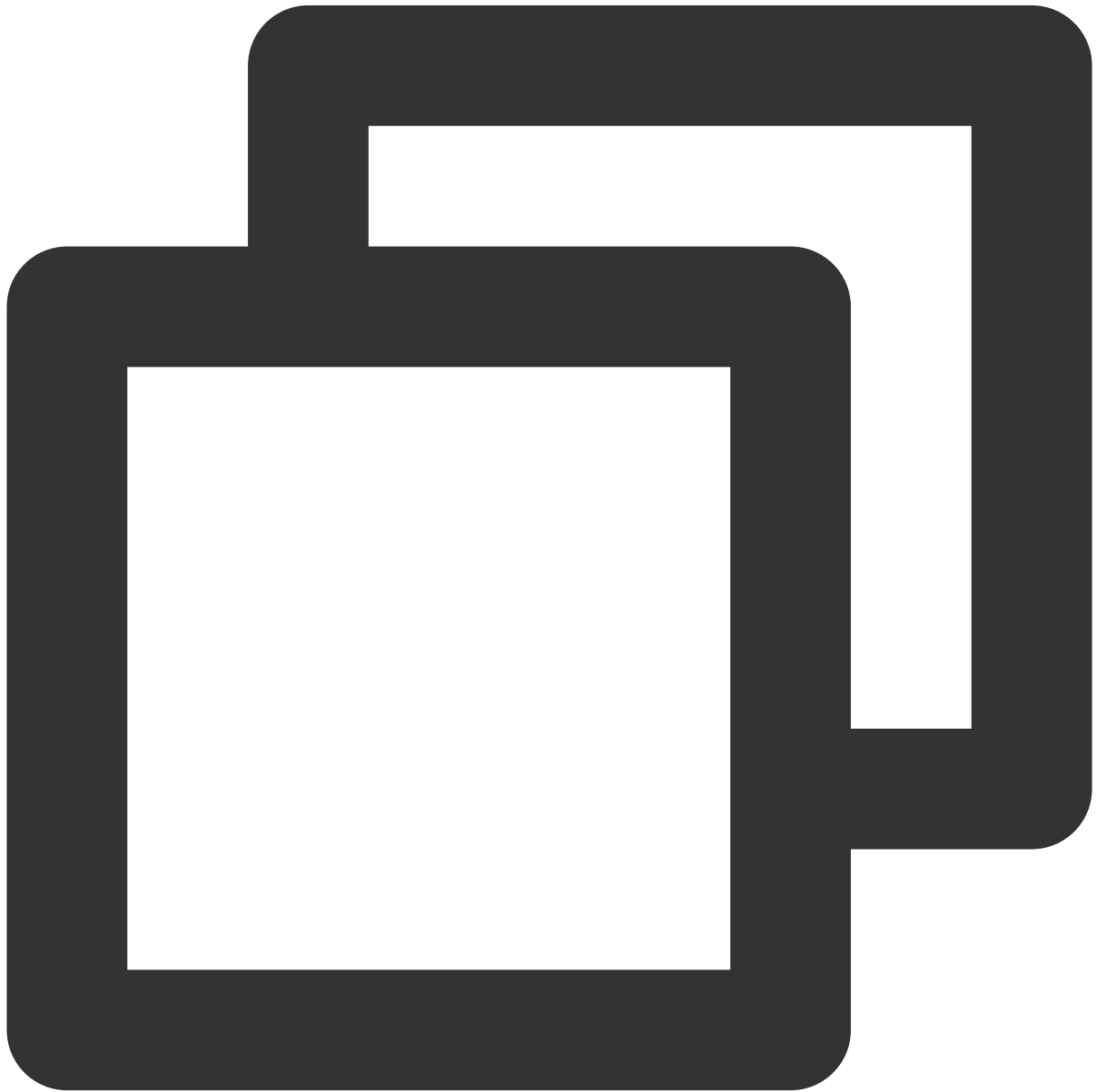
```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';
```

```
V2TimCallback res = await TencentImSDKPlugin.v2TIMManager.login (
    userID: userID,
    userSig: userSig,
);
```

Note:

The test account is for development and testing only. Before the application is launched, the correct method for generating a UserSig is to integrate the UserSig calculation code into your server and provide an application-oriented API. When UserSig is needed, your application can send a request to the business server for a dynamic UserSig. For more information, see Generating UserSig.

Sending a message

Detailed documentation

The following shows how to send a text message:

1. Call `createTextMessage(String)` to create a text message.

2. Get the message ID from the returned value.

3. Call `sendMessage()` and pass in the `id` , while `receiver` can be the ID of the other test account created earlier, `groupID` is not required for sending a one-to-one message.

Sample code:
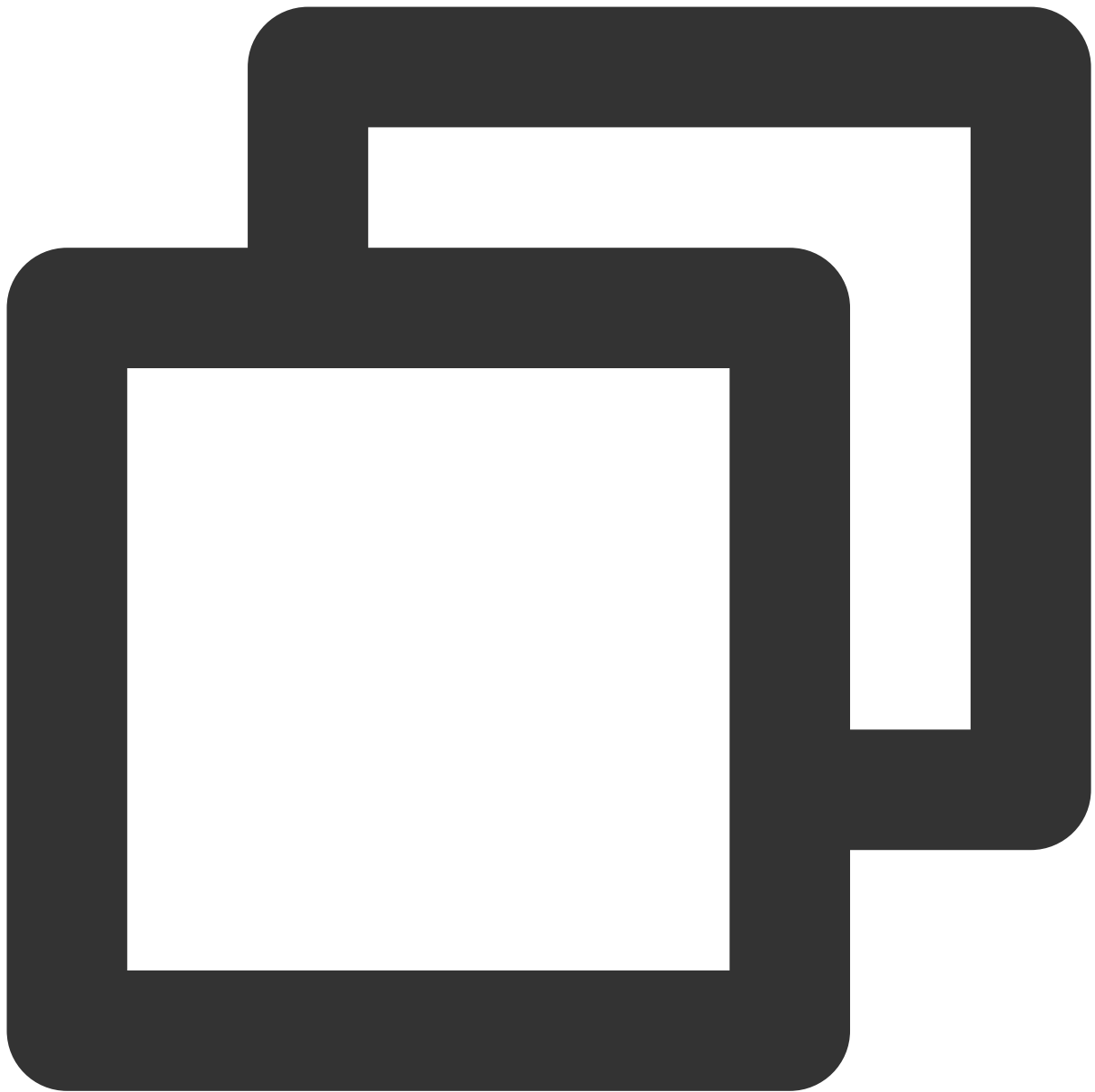
```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';

V2TimValueCallback<V2TimMsgCreateInfoResult> createMessage =
        await TencentImSDKPlugin.v2TIMManager
            .getMessageManager()
            .createTextMessage(text: "The text to create");

String id = createMessage.data!.id!; // The message creation ID

V2TimValueCallback<V2TimMessage> res = await TencentImSDKPlugin.v2TIMManager
        .getMessageManager()
```

Tencent Cloud

```
    .sendMessage(
        id: id, // Pass in the message creation ID to
        receiver: "The userID of the destination user",
        groupID: "The groupID of the destination group",
        );
```

Note:

If sending fails, it may be that your `sdkAppID` doesn't support sending messages to strangers. In this case, you can disable the relationship check feature in the console.

Disable the friend relationship chain check here.

Obtaining the conversation list

Detailed documentation

Log in with the second test account to load the conversation list.

The conversation list can be obtained in two ways:

1. Listen for the persistent connection callback to update the conversation list in real time.

2. Call an API to get the conversation list at certain time points.

Common use cases include:

Getting the conversation list when the application starts and listening for the persistent connection callback to update the conversation list in real time.

Requesting the conversation list at certain time points

To get the conversation list, you need to maintain `nextSeq` and record its current position.
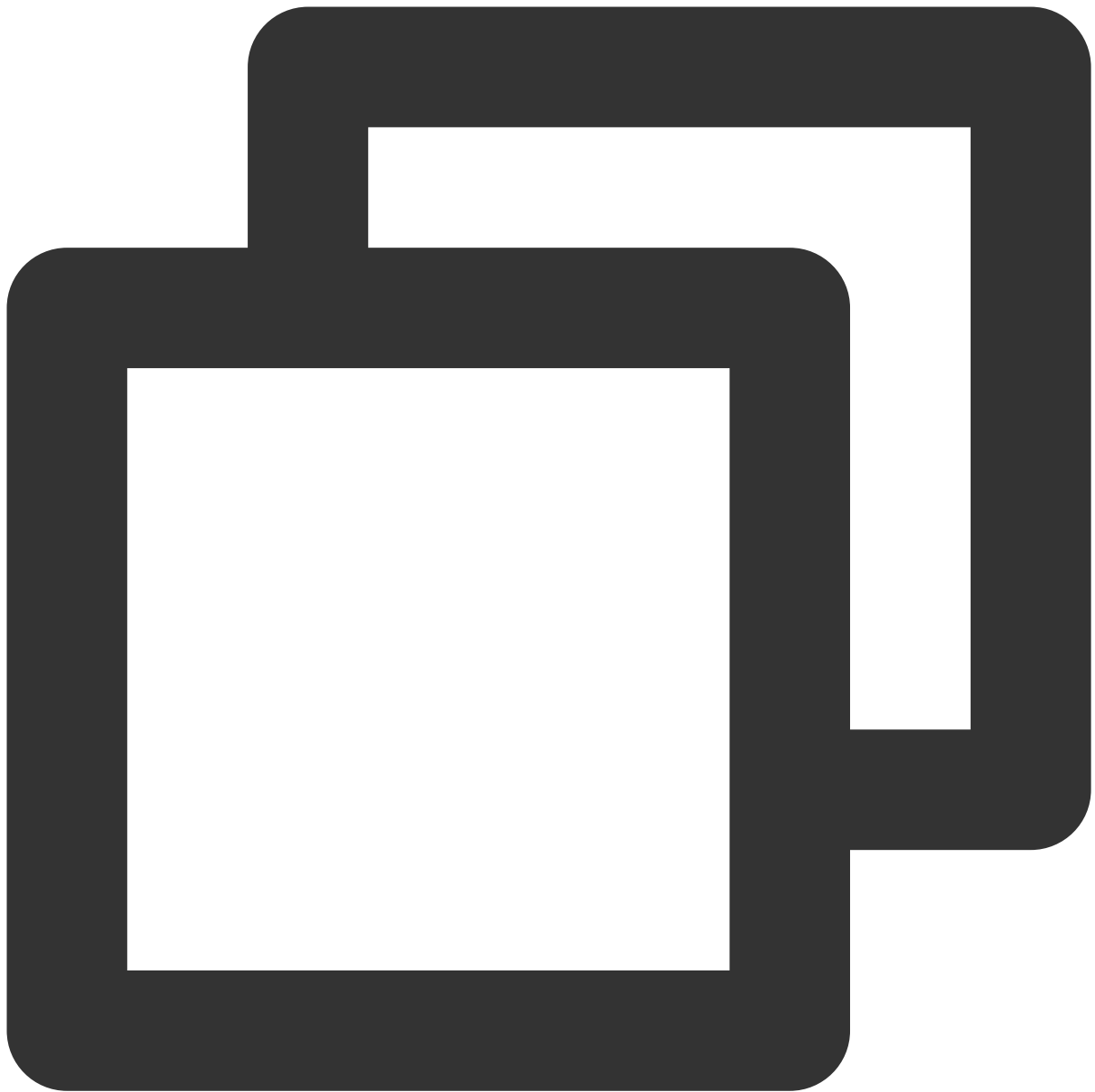
```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';

String nextSeq = "0";

getConversationList() async {
  V2TimValueCallback<V2TimConversationResult> res = await TencentImSDKPlugin
      .v2TIMManager
      .getConversationManager()
      .getConversationList(nextSeq: nextSeq, count: 10);

  nextSeq = res.data?.nextSeq ?? "0";
```

```
}
```

At this point, you can see the message sent by the first test account in the previous step.

Listening for the persistent connection to get the conversation list in real time

Mount the conversation list listener, process the callback event, and update the UI.

1. Mount the listener.
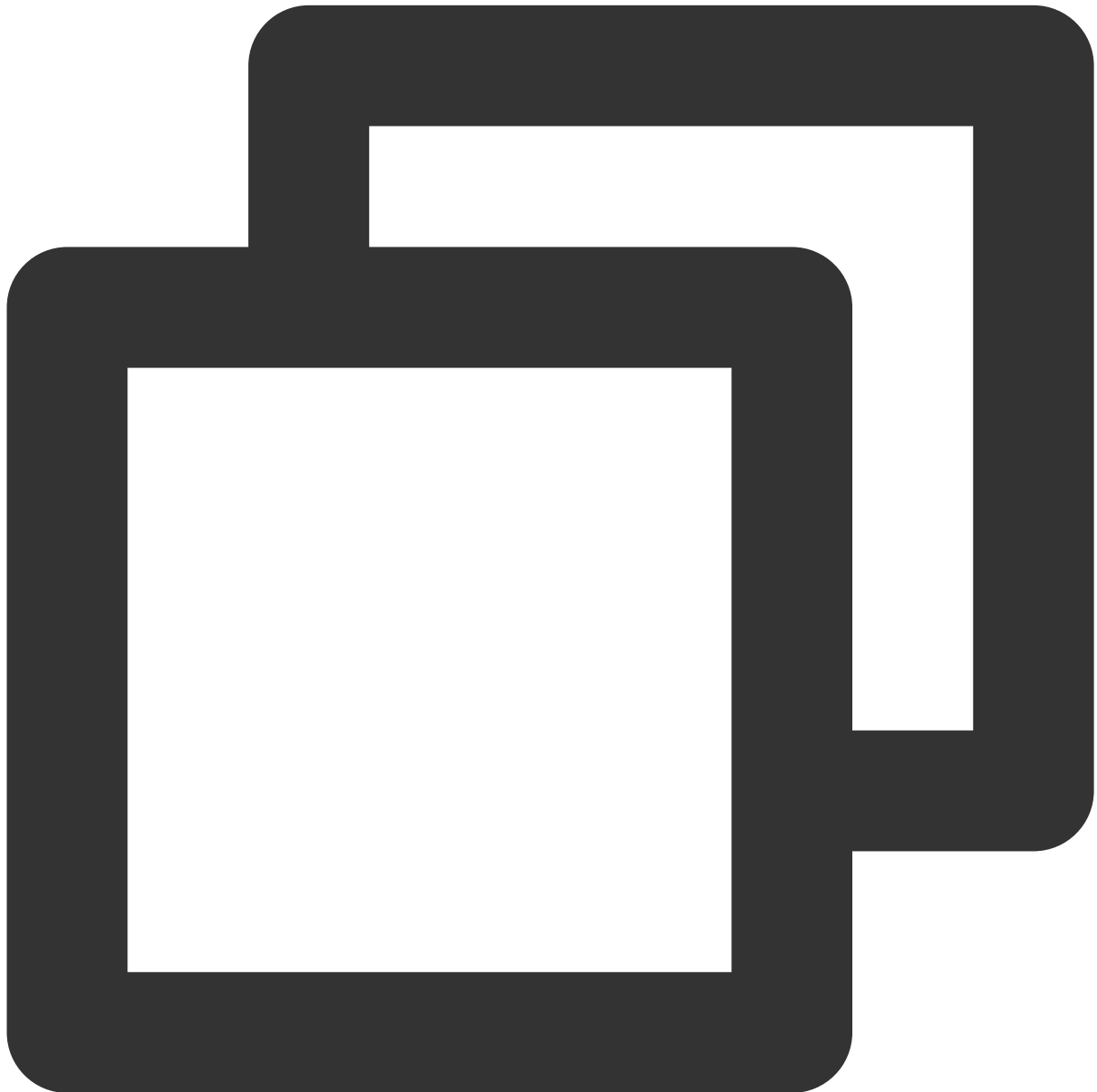

```
await TencentImSDKPlugin.v2TIMManager
  .getConversationManager()
  .setConversationListener(
```

```
    listener: new V2TimConversationListener(
       onConversationChanged: (List<V2TimConversation> list){
          _onConversationListChanged(list);
  },
       onNewConversation:(List<V2TimConversation> list){
          _onConversationListChanged(list);
  },
```

2. Process the callback event and display the latest conversation list on the UI.
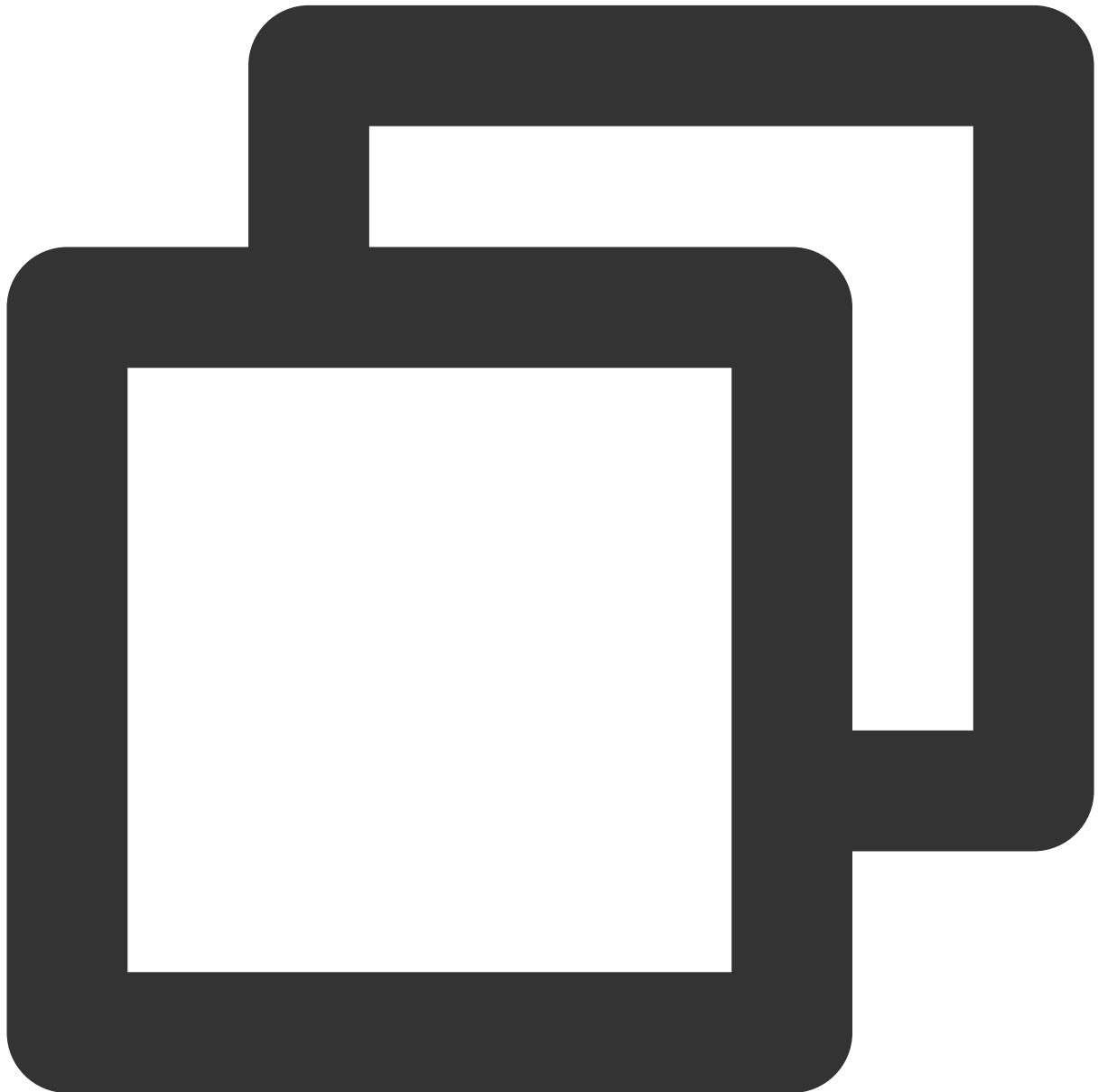

```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';
```

```
List<V2TimConversation> _conversationList = [];

_onConversationListChanged(List<V2TimConversation> list) {
  for (int element = 0; element < list.length; element++) {
int index = _conversationList.indexWhere(
    (item) => item!.conversationID == list[element].conversationID);
if (index > -1) {
_conversationList.setAll(index, [list[element]]);
} else {
_conversationList.add(list[element]);
}
  }
```

Receiving messages

[Detailed documentation](#)

There are two methods to receive messages in the Chat SDK for Flutter:

1. Listen for the persistent connection callback to get message changes and update and render the historical message list in real time.

2. Call an API to load the message history at certain time points.

   Common use cases include:

1. Requesting a certain number of historical messages to be displayed when a new conversation is opened in the UI.

2. Listening for the persistent connection callback to receive messages in real time and adding them to the historical message list.

Requesting the historical message list at certain time points

To avoid affecting the loading speed, we recommend you limit the number of messages loaded to 20 per page.

You need to dynamically record the current number of pages for the next request.

Sample code:
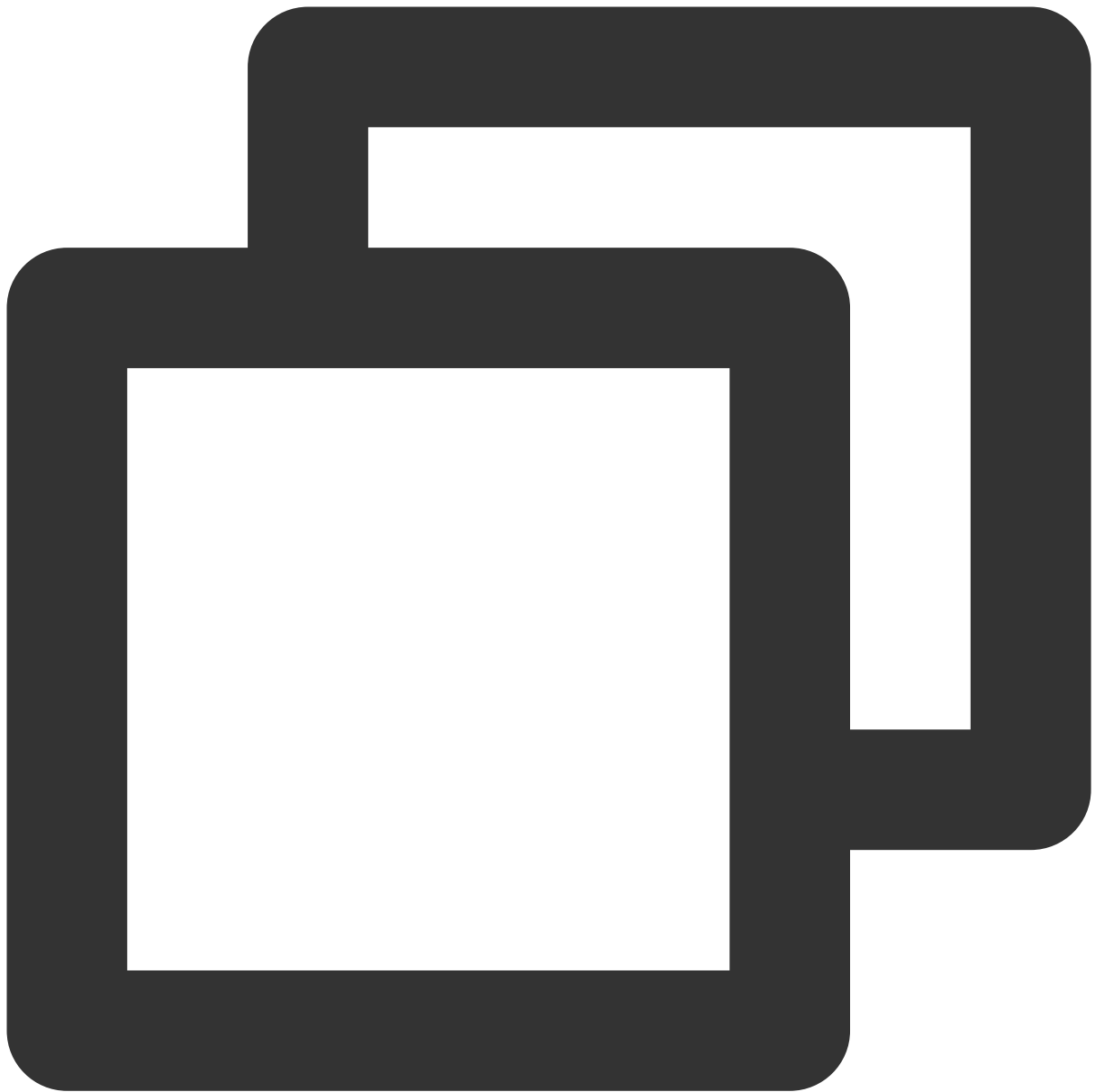
```
import 'package:tencent_cloud_chat_sdk/tencent_cloud_chat_sdk.dart';

  V2TimValueCallback<List<V2TimMessage>> res = await TencentImSDKPlugin
      .v2TIMManager
      .getMessageManager()
      .getGroupHistoryMessageList(
        groupID: "groupID",
        count: 20,
        lastMsgID: "",
      );
```

```
    List<V2TimMessage> msgList = res.data ?? [];

    // here you can use msgList to render your message list
      }
```

Listening for the persistent connection callback to get new messages in real time

After the historical message list is initialized, new messages are from the persistent connection

`V2TimAdvancedMsgListener.onRecvNewMessage` .

After the `onRecvNewMessage` callback is triggered, you can add new messages to the historical message list as needed.

Sample code for binding a listener:
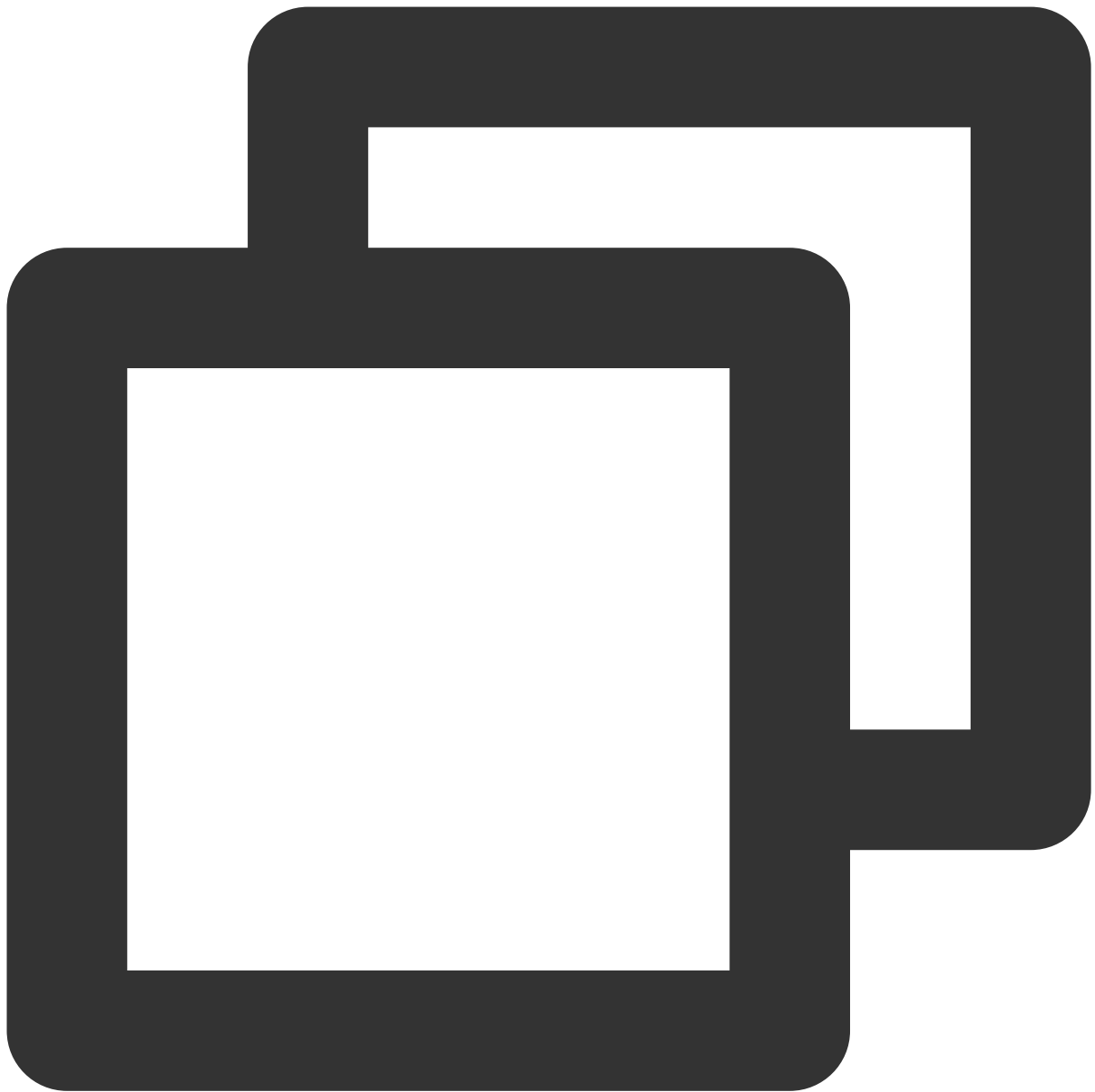
Tencent Cloud


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

final adVancesMsgListener = V2TimAdvancedMsgListener(
onRecvNewMessage: (V2TimMessage newMsg) {
  _onReceiveNewMsg(newMsg);
},
/// ... other listeners related to message
);

TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
```

```
        .addAdvancedMsgListener(listener: adVancesMsgListener);
```

At this point, you have completed the Chat module development, and now users can send and receive messages and enter different conversations.

You can develop more features, such as group, user profile, relationship chain, offline push, and local search.

For more information, see Integration Solution (No UI).

Expanding to More Platforms

Tencent Cloud Chat for Flutter SDKs support Android, iOS and Windows platforms by default. You can also expand to more platforms (web and macOS).

Web

To enable support for web, you need to perform the following extra steps in addition to those for enabling support for Android and iOS:

Upgrading to Flutter 3.x

Flutter 3.x has been dramatically optimized for web performance and is highly recommended for Flutter web project development.

Importing JS

Note:

If your existing Flutter project does not support web, run `flutter create .` in the root directory of the project to add web support.

Go to the `web/` directory of your project and run `npm` or `yarn` to install relevant JS dependencies. Initialize the project as instructed.
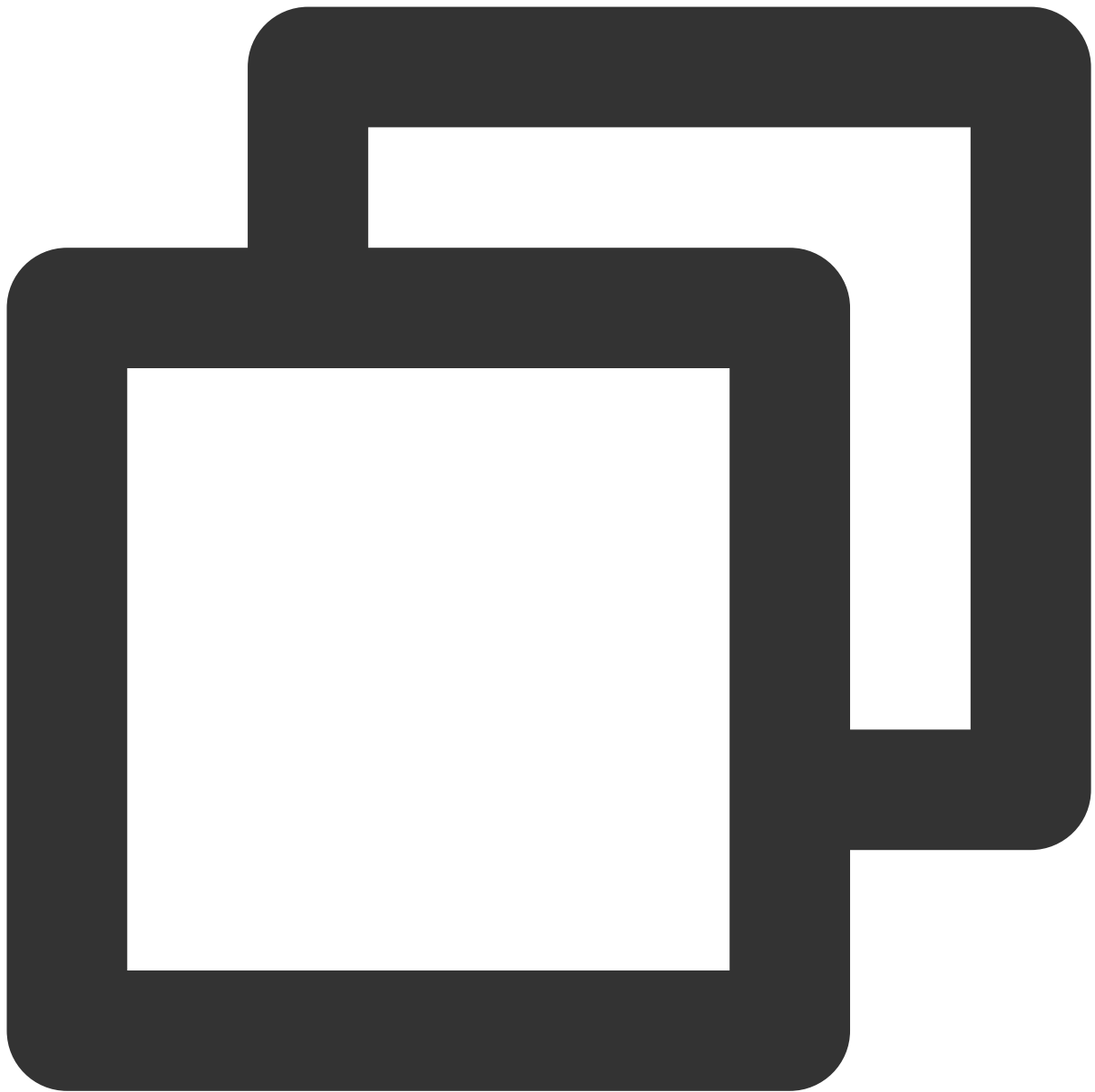
```
cd web

npm init

npm i @tencentcloud/chat

npm i tim-upload-plugin
```

Open `web/index.html` and import the JS files in `<head> </head>` . See below:

```
<script src="./node_modules/tim-upload-plugin/index.js"></script>
<script src="./node_modules/@tencentcloud/chat/index.js"></script>
<script src="./node_modules/@tencentcloud/chat/modules/group-module.js"></script>
<script src="./node_modules/@tencentcloud/chat/modules/relationship-module.js"></sc
<script src="./node_modules/@tencentcloud/chat/modules/signaling-module.js"></scrip
```
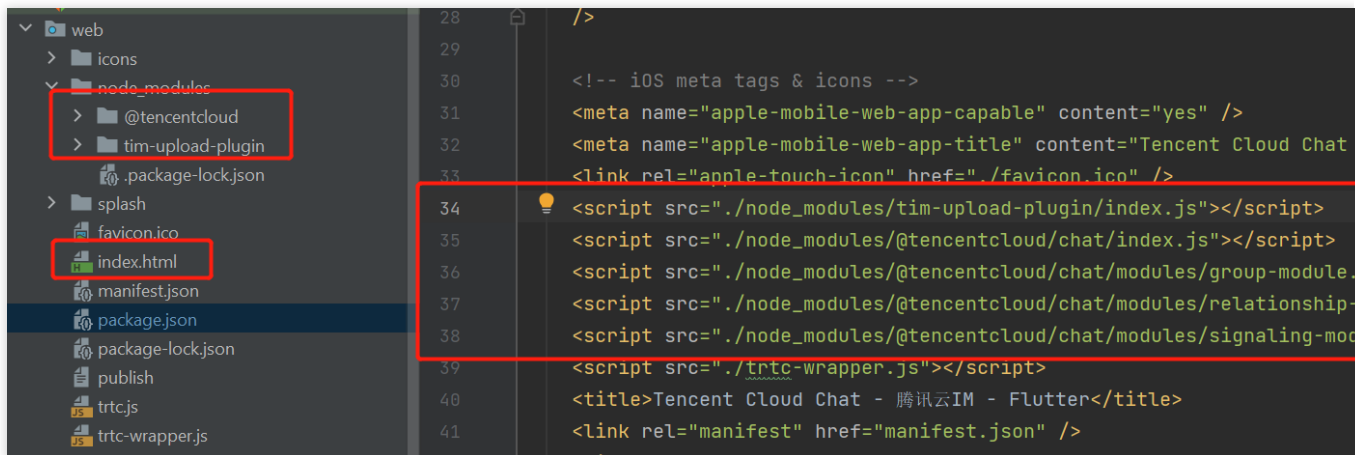
macOS

Additional configurations are required for the macOS platform. Follow the steps below to configure the macOS platform:

1. Open the `macos/Runner/DebugProfile.entitlements` and `macos/Runner/Release.entitlements` files in your project.

2. Add the following lines to each file:
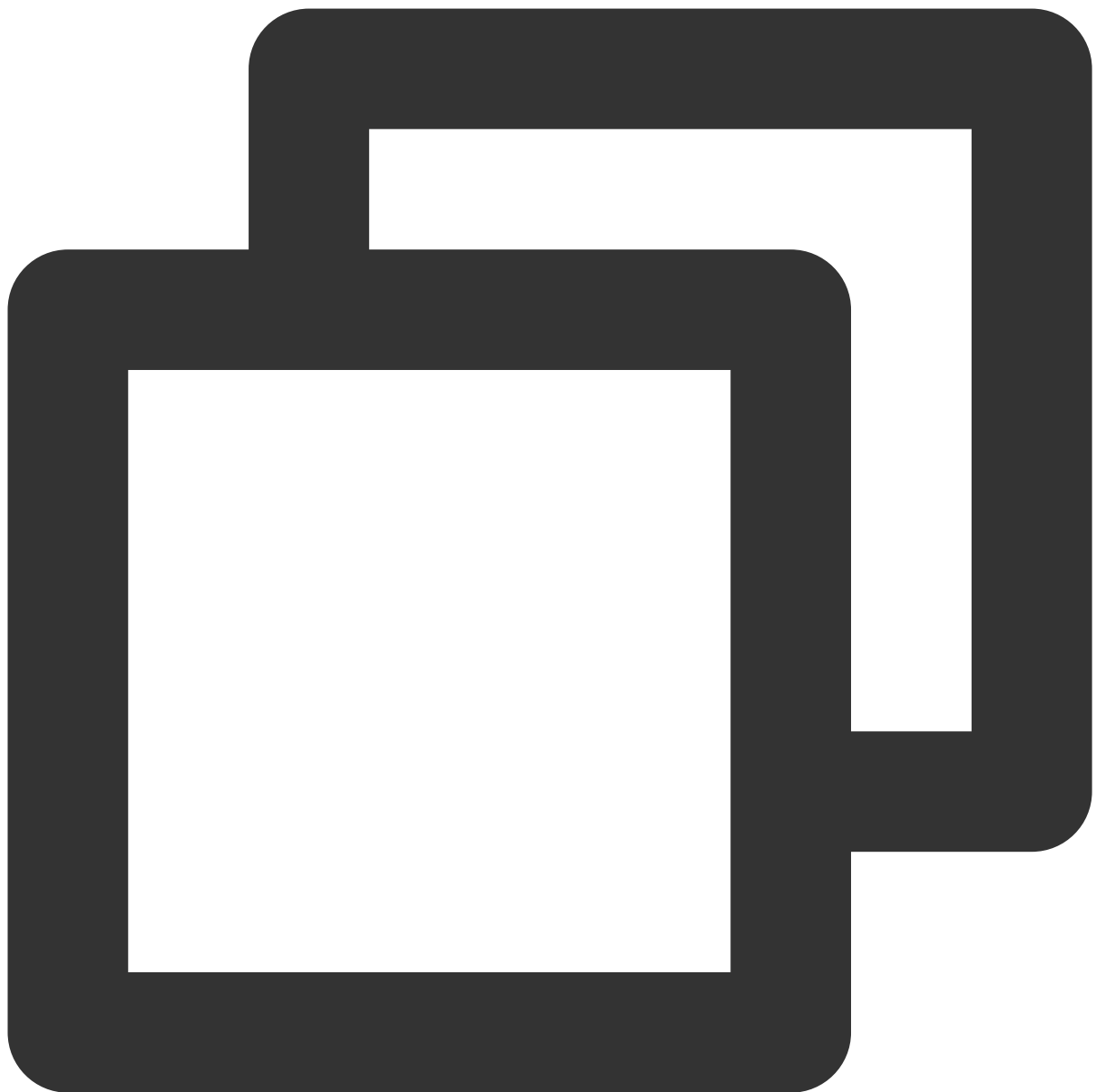
```
<key>com.apple.security.network.client</key>
<true/>
```
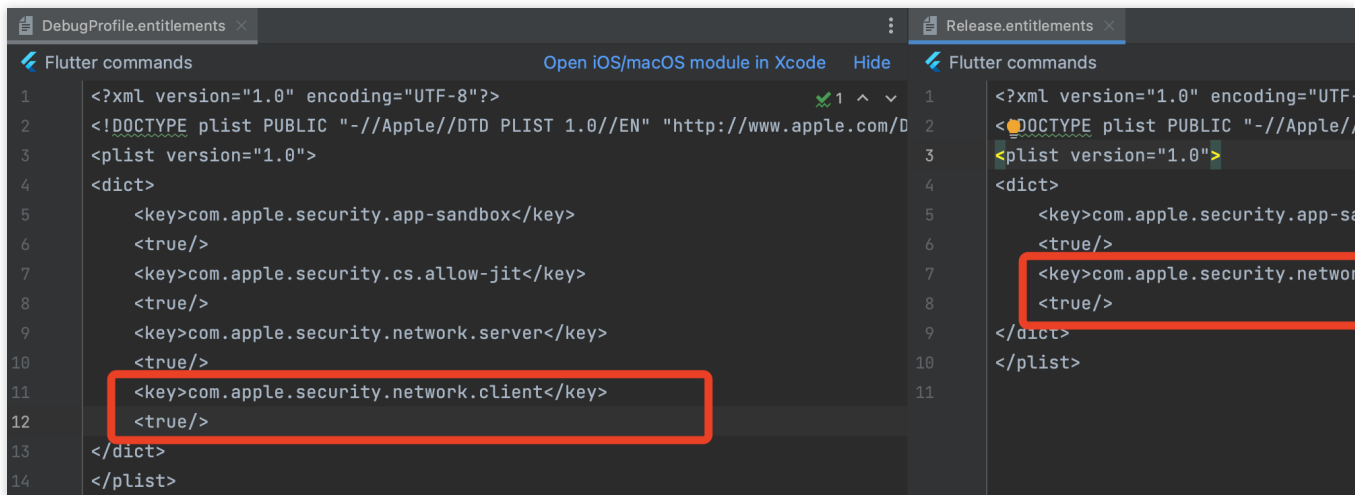
These lines grant your app the necessary permissions to access the network as a client.

This configuration is essential for ensuring proper communication between your app and the backend services on the macOS platform.

FAQs

What should I do if Pods dependency installation fails in iOS devices?

Solution 1: If an error occurs after the configuration, click **Product** > **Clean Build Folder**, clear the product, and run `pod install` or `flutter run` again.

Solution 2: Manually delete the `ios/Pods` folder and the `ios/Podfile.lock` file and run the following command to reinstall the dependencies

```
cd ios
sudo gem install ffi
pod install --repo-update
```

Flutter environment

If you want to check the Flutter environment, run `Flutter doctor` to detect whether the Flutter environment is ready.

What should I do if an error occurs when I run the Demo on a Windows platform?
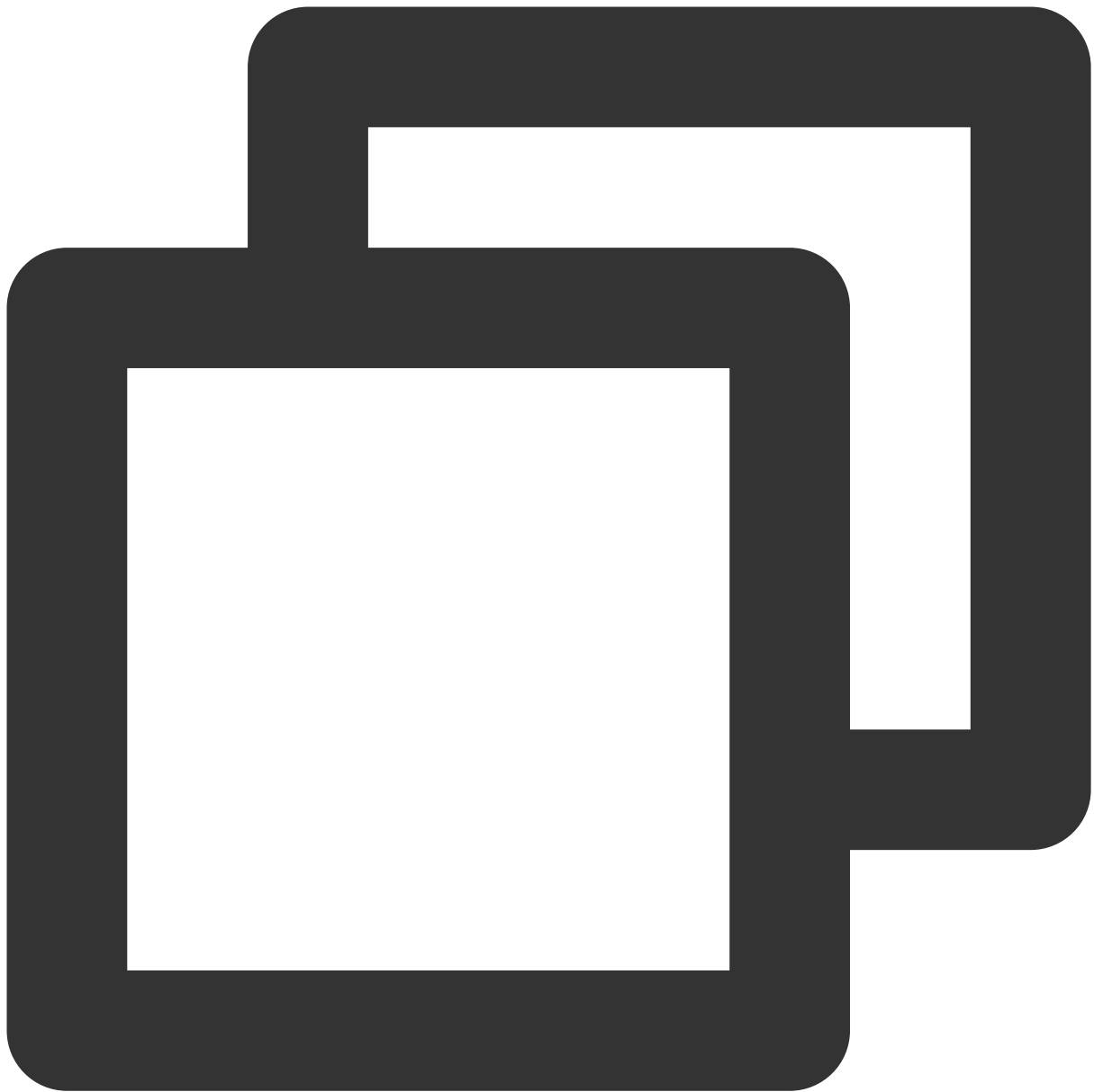
```
Nuget.exe not found, trying to download or use cached version.
```

This error message indicates that `nuget.exe` was not found on your system, so the program is attempting to download or use a cached one. `Nuget.exe` is a command-line utility for the NuGet Package Manager and is used to manage dependencies in `.NET` projects.

To fix this issue, you can manually download and install `nuget.exe` by following these steps:

1. Go to the download page on the NuGet official website: https://www.nuget.org/downloads

2. Find the Windows x86 Commandline section and download `nuget.exe` as needed.

3. Save the downloaded `nuget.exe` file to an appropriate location, for example, `C:\\NuGet`.

4. Add the folder where you placed `nuget.exe` to your `PATH` environment variable to make the `nuget.exe` globally available in the command line.

How do I query error codes?

For Chat SDK API error codes, see here.

For the UIKit errors and specific User Notification event calls, see here.

React Native

Last updated：2024-04-11 15:04:18

This document describes how to quickly run the Tencent Cloud Chat demo for React Native.

Environment Requirements

Platform	Version
React Native	v0.63.4 or later
Android	Android Studio 3.5 or later; devices with Android 4.1 or later for apps
iOS	Xcode 11.0 or later. Ensure that your project has a valid developer signature.

Prerequisites

You have signed up for a Tencent Cloud account and completed identity verification.

Part 1. Creating Test Accounts

In the Chat console, select your application and click **Auxiliary Tools** > **UserSig Generation & Verification** on the left sidebar. Enter two UserIDs and generate UserSig values, and copy the `UserID`, `Key`, and `UserSig` for subsequent logins.

Note:

 This account is for development and testing only. Before the application is launched, the correct `UserSig` distribution method is to use the server to generate `UserSig` and provide an application-oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig`. For more information, see Generating UserSig.
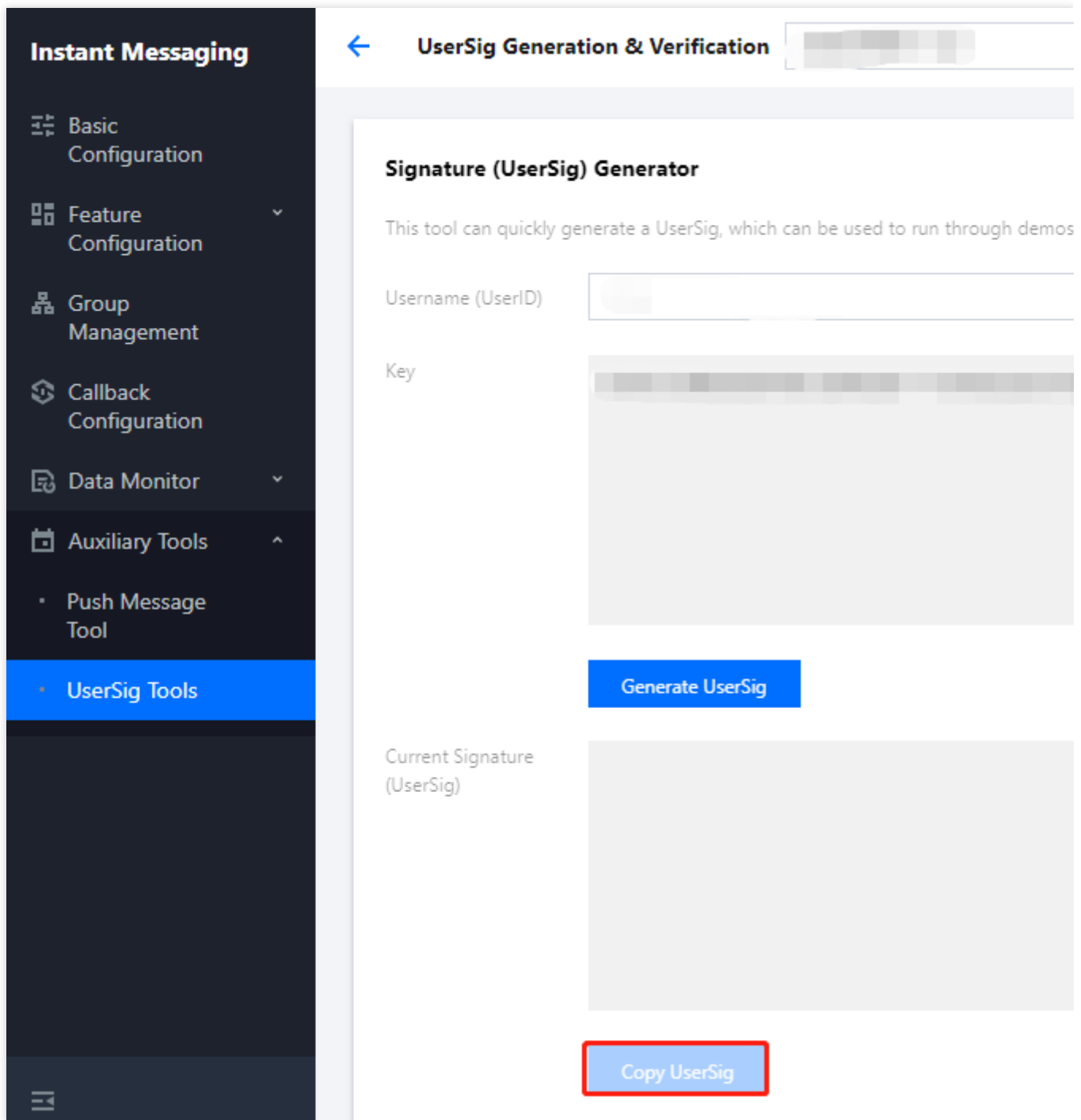
Part 2. Integrating the SDK for React Native

Prerequisites

You have created or already have a React Native project.

Directions

Installing the Chat SDK

Run the following command to install the latest version of the Chat SDK for React Native:
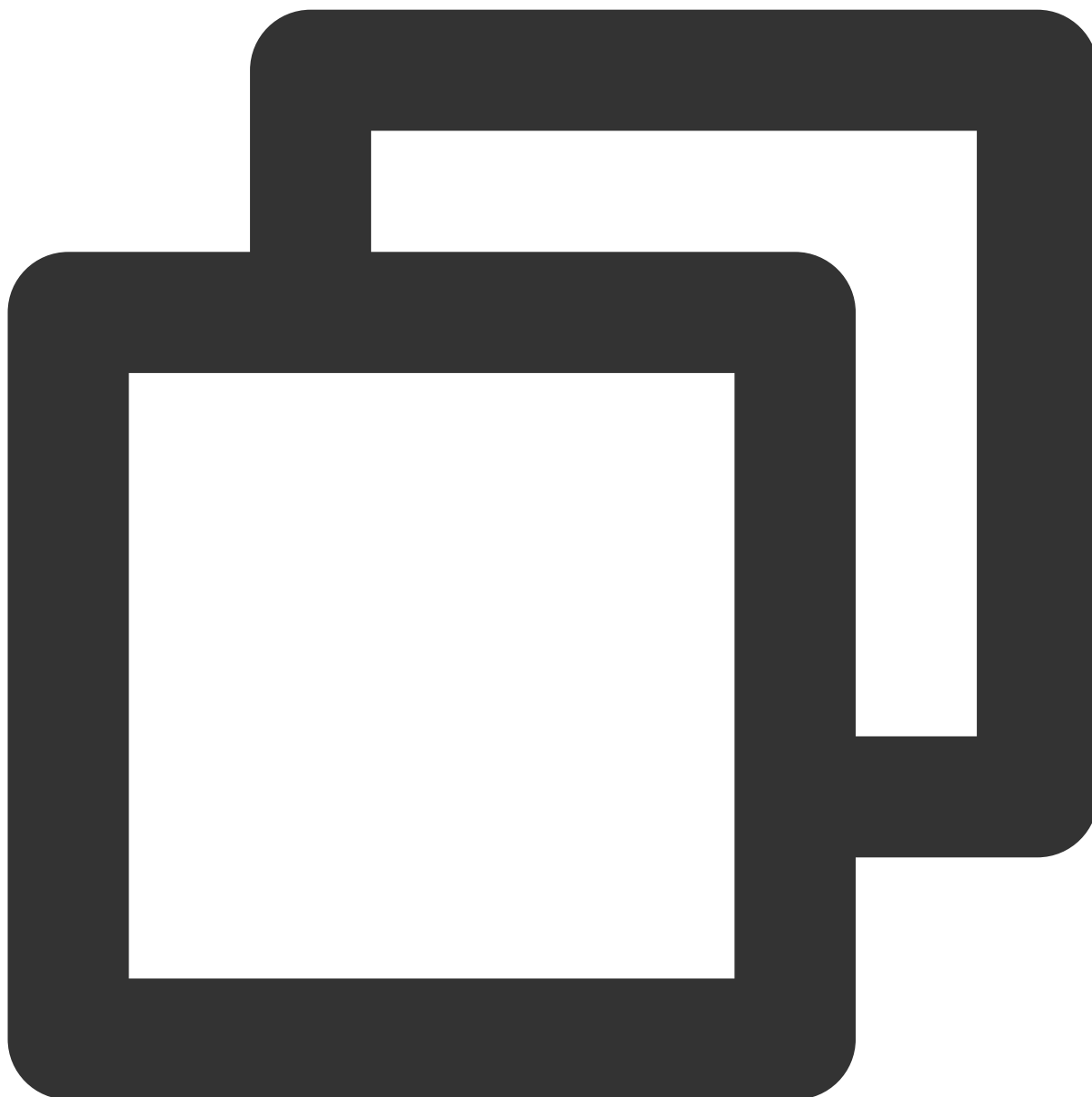
```
// npm
npm install react-native-tim-js

// yarn
yarn add react-native-tim-js
```

Initializing the SDK

Call `initSDK` to initialize the SDK and pass in your `sdkAppID`.
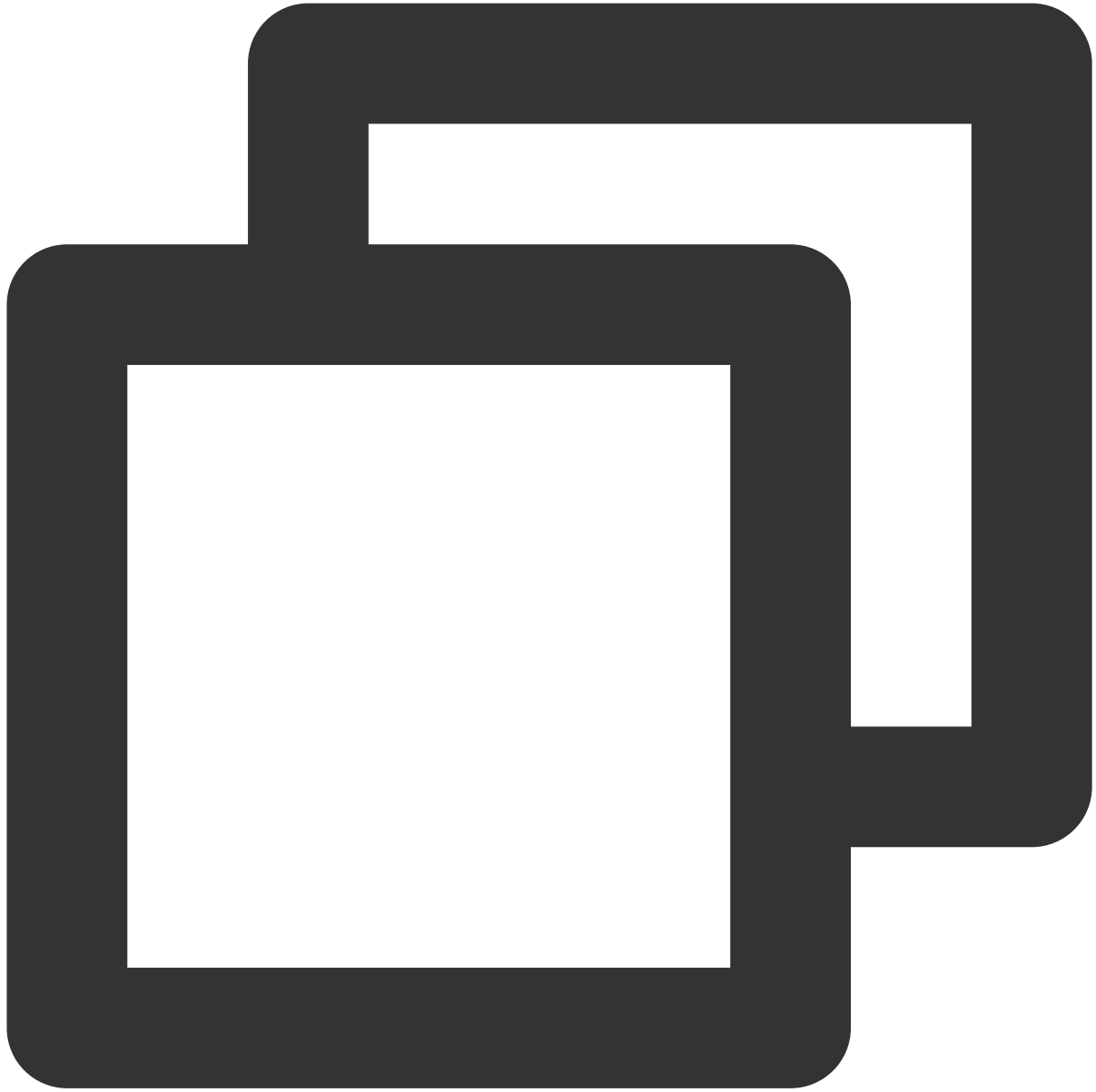

```
import { TencentImSDKPlugin, LogLevelEnum } from 'react-native-tim-js';

const sdkAppID = 0;  // Replace 0 with the SDKAppID of your Chat application when i
const loglevel = LogLevelEnum.V2TIM_LOG_DEBUG; // Log
const listener = {
    onConnecting: () => {
    },
    //...
}
```

```
TencentImSDKPlugin.v2TIMManager.initSDK(
    sdkAppID,
    loglevel,
    listener,
);
```

In this step, you can mount some listeners to the Chat SDK, mainly including those for network status and user information change. For more information, see Interface V2TimSDKListener.Log in with a test account initially generated in the console for verification.

Logging in with a test account

Call `TencentImSDKPlugin.v2TIMManager.login` to log in with one of the test accounts you created. If the returned `res.code` is 0, login is successful.
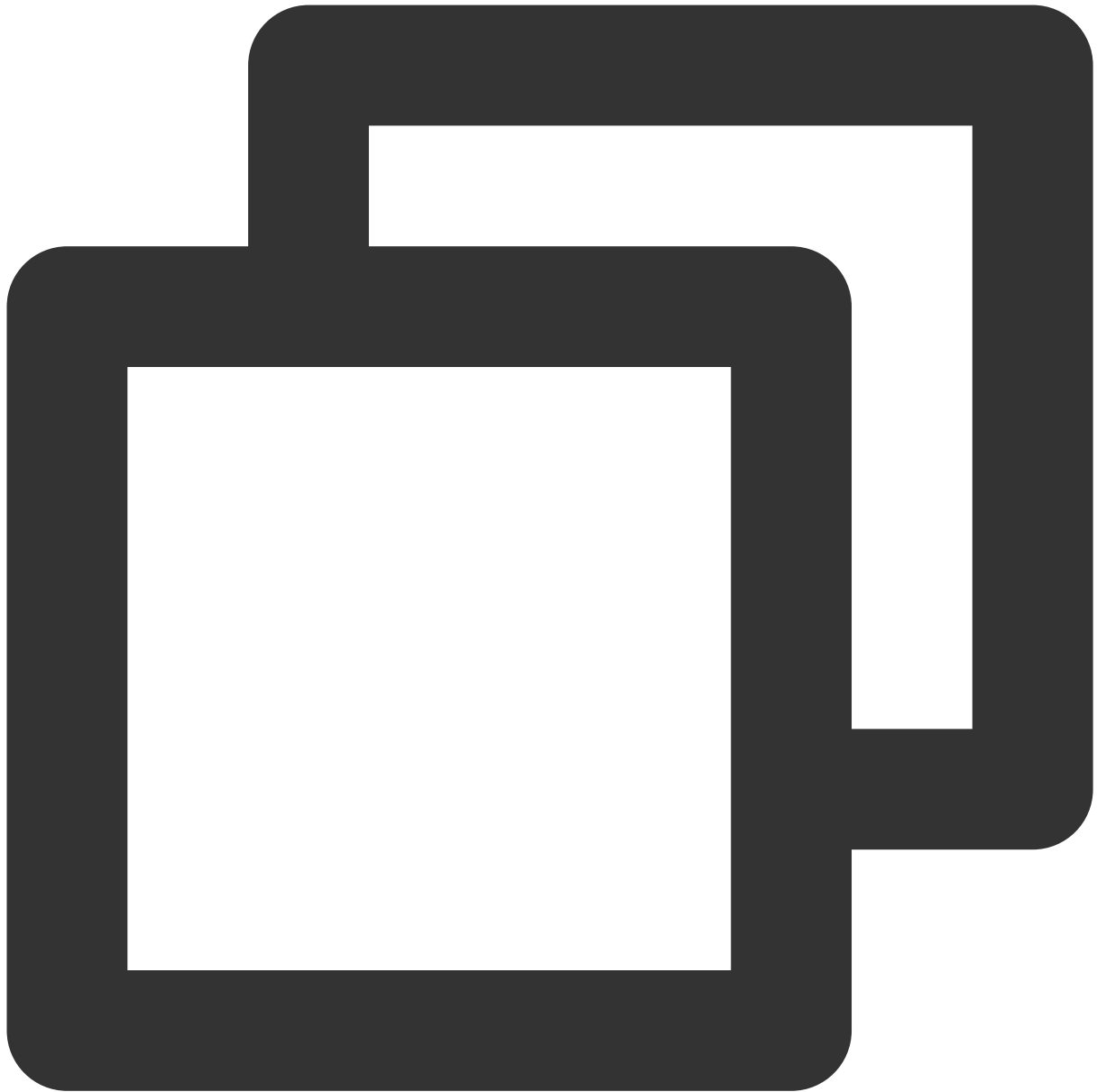
```
import { TencentImSDKPlugin } from 'react-native-tim-js';
 const res = await TencentImSDKPlugin.v2TIMManager.login(
 userID: userID,
 userSig: userSig,
  );
```

Note:

This account is for development and testing only. Before the application is launched, the correct `UserSig`

distribution method is to integrate the calculation code of `UserSig` into your server and provide an application-

oriented API. When `UserSig` is needed, your application can send a request to the business server for a dynamic `UserSig` . For more information, see Generating UserSig.

Sending messages

The following shows how to send a text message:

1. Call `createTextMessage(String)` to create a text message.

2. Get the message ID from the returned value.

3. Call `sendMessage()` and pass in the message ID. For `receiver` , you can pass in the ID of the other test account. `groupID` can be left empty.

Sample code:
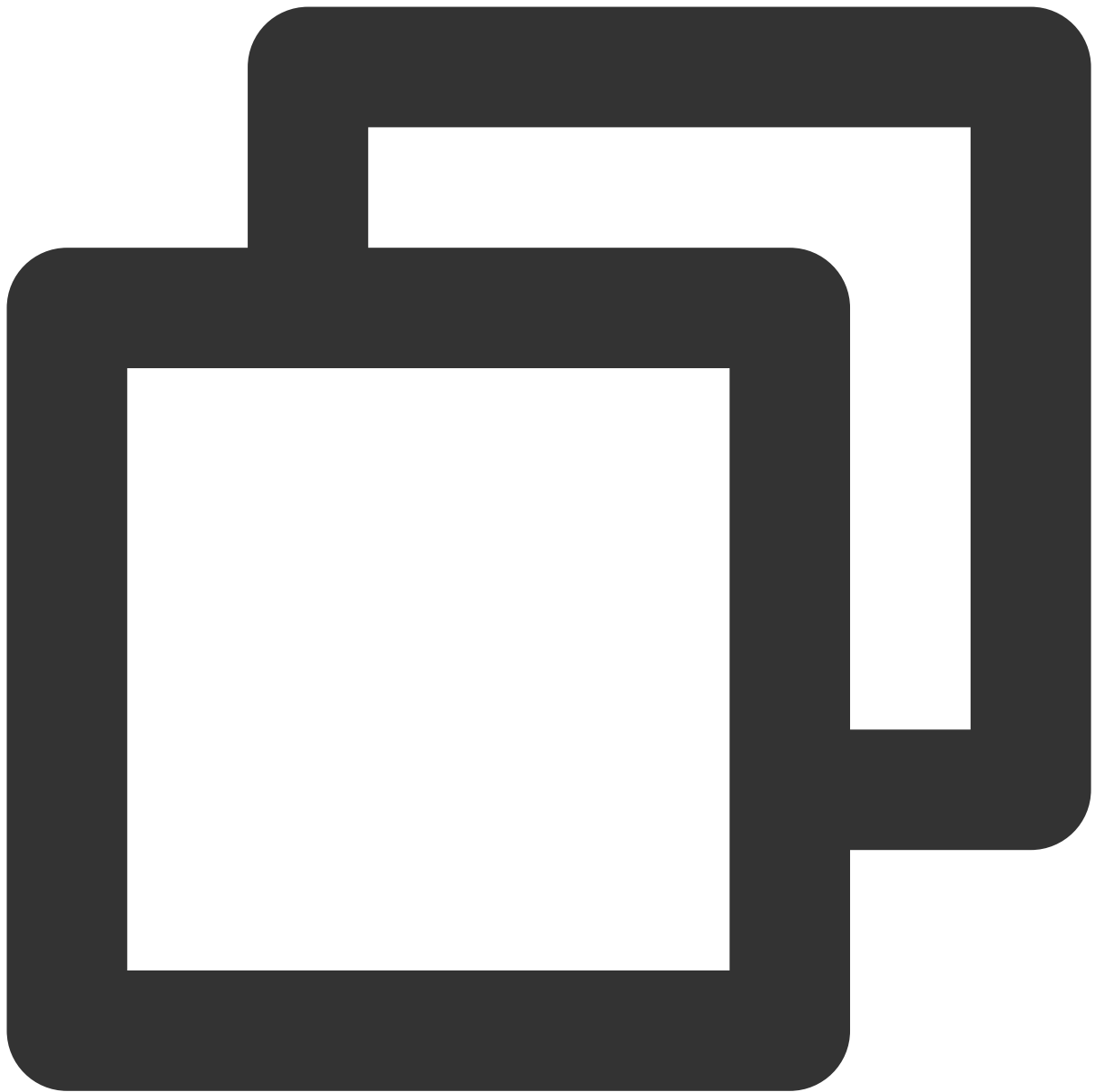
```
import { TencentImSDKPlugin } from 'react-native-tim-js';

const createMessage =
      await TencentImSDKPlugin.v2TIMManager
          .getMessageManager()
          .createTextMessage("The text to create");

const id = createMessage.data!.id!; // The message creation ID

const res = await TencentImSDKPlugin.v2TIMManager
      .getMessageManager()
```

```
    .sendMessage({
        id: id, // Pass in the message creation ID
        receiver: "The userID of the destination user",
        groupID: "The groupID of the destination group",
    });
```

Note:

If sending fails, it may be that your `sdkAppID` doesn't support sending messages to strangers. In this case, you can disable the relationship check feature in the console.

Obtaining the conversation list

Log in with the other test account to pull the conversation list.

The conversation list can be obtained in two ways:

1. Listen for the persistent connection callback to update the conversation list in real time.

2. Call an API to get the conversation list at certain time points.

Common use cases include:

Get the conversation list upon application start and listen for the persistent connection to update the conversation list in real time.

Requesting the conversation at certain time points

To get the conversation list, you need to maintain `nextSeq` and record its current position.
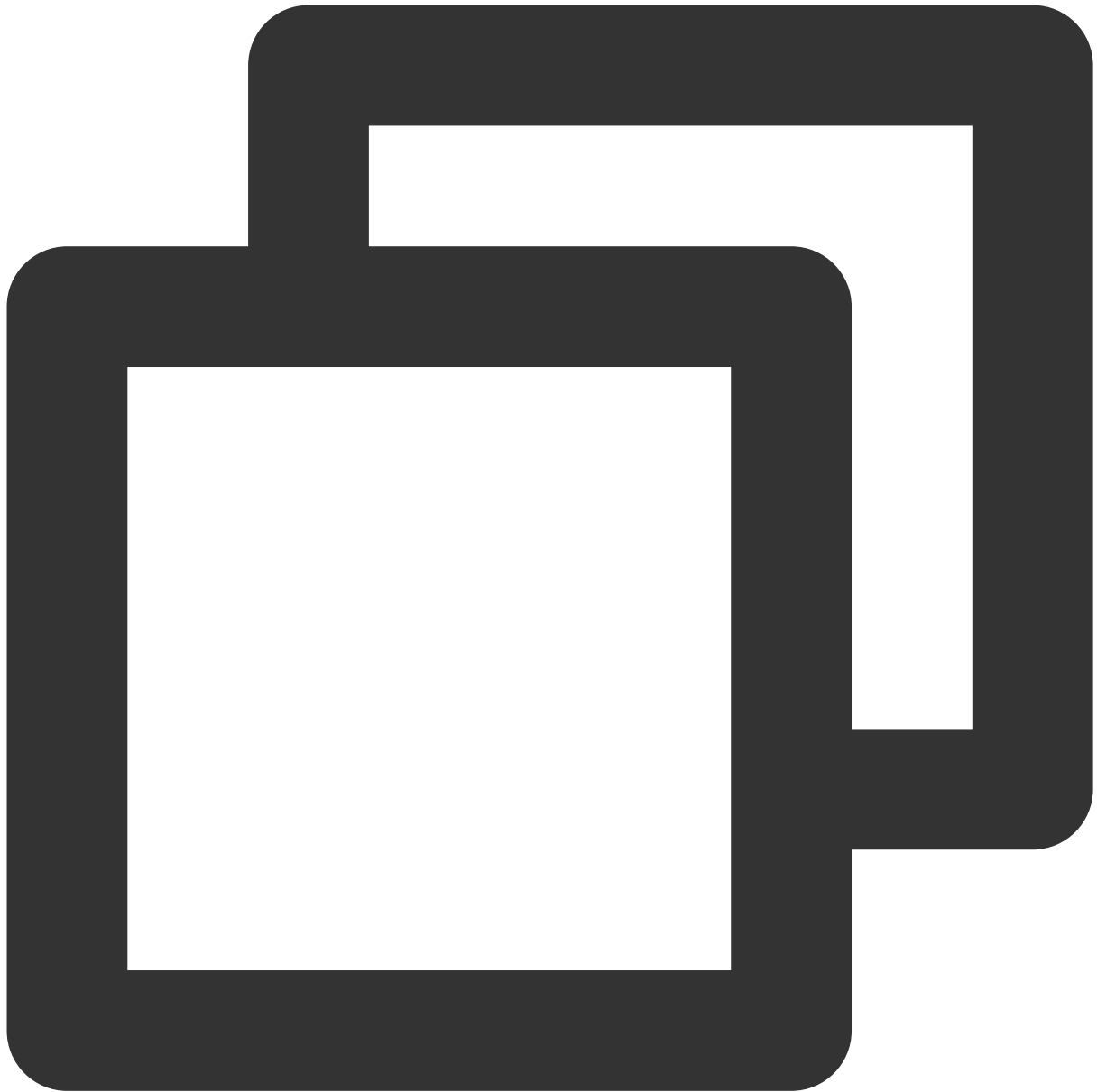
```
import { useState } from "react";
import { TencentImSDKPlugin } from "react-native-tim-js";

const [nextSeq, setNextSeq] = useState<string>("0");

const getConversationList = async () => {
  const count = 10;
  const res = await TencentImSDKPlugin.v2TIMManager
    .getConversationManager()
    .getConversationList(count, nextSeq);
  setNextSeq(res.data?.nextSeq ?? "0");
```

```
};
```

At this point, you can see the message sent by the other test account in the previous step.

Listening for the persistent connection to get the conversation list in real time

Mount listeners to the SDK, process the callback event, and update the UI.

1. Mount listeners.
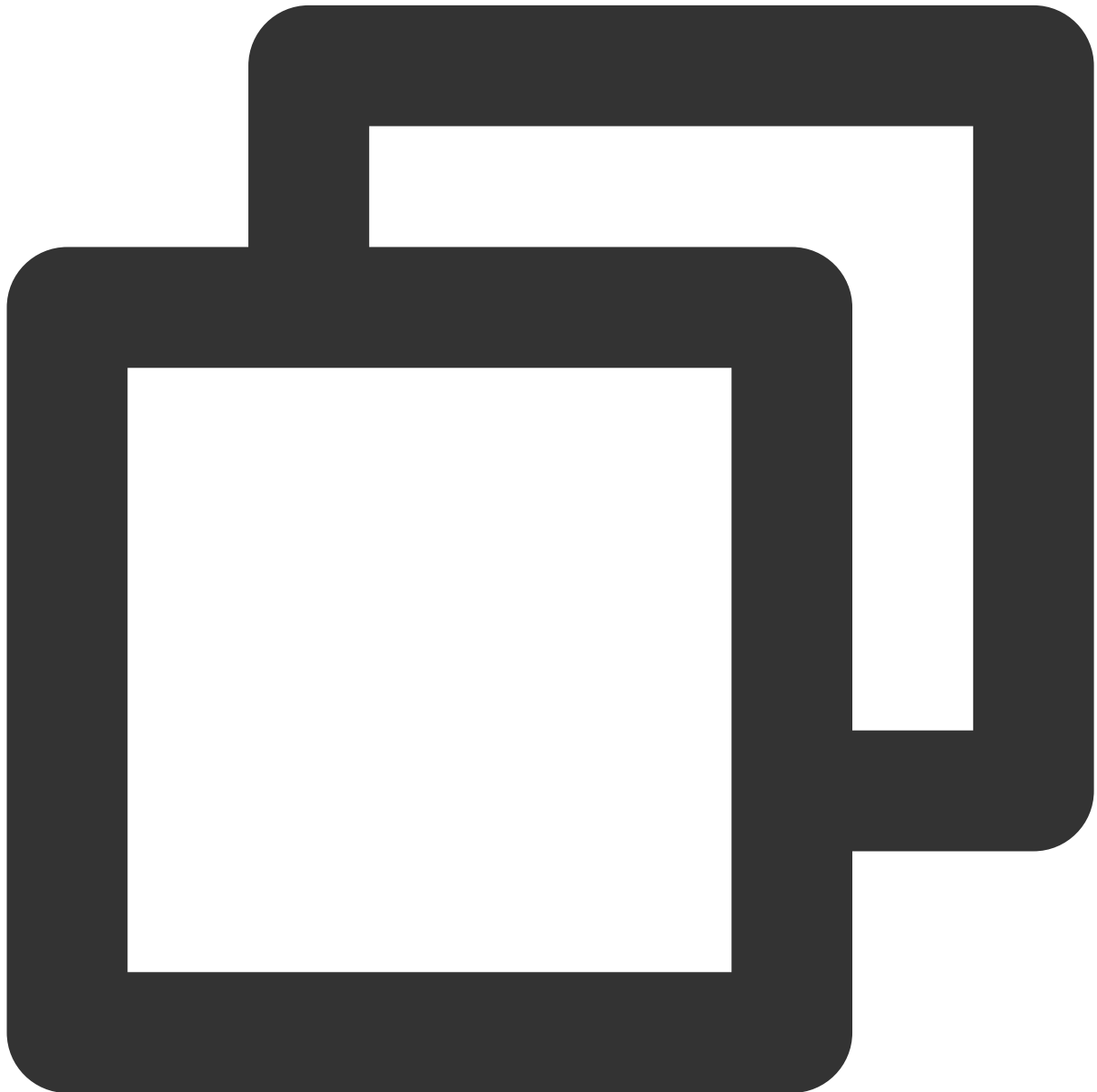
```
import { TencentImSDKPlugin } from "react-native-tim-js";

const addConversationListener = () => {
```

```
  TencentImSDKPlugin.v2TIMManager
.getConversationManager()
.addConversationListener({
  onNewConversation: (conversationList) => {
    // new conversation created callback
    _onConversationListChanged(conversationList);
  },
  onConversationChanged: (conversationList) => {
    // conversation changed callback
    _onConversationListChanged(conversationList);
  },
});
};
```

2. Process the callback event and display the latest conversation list on the UI.
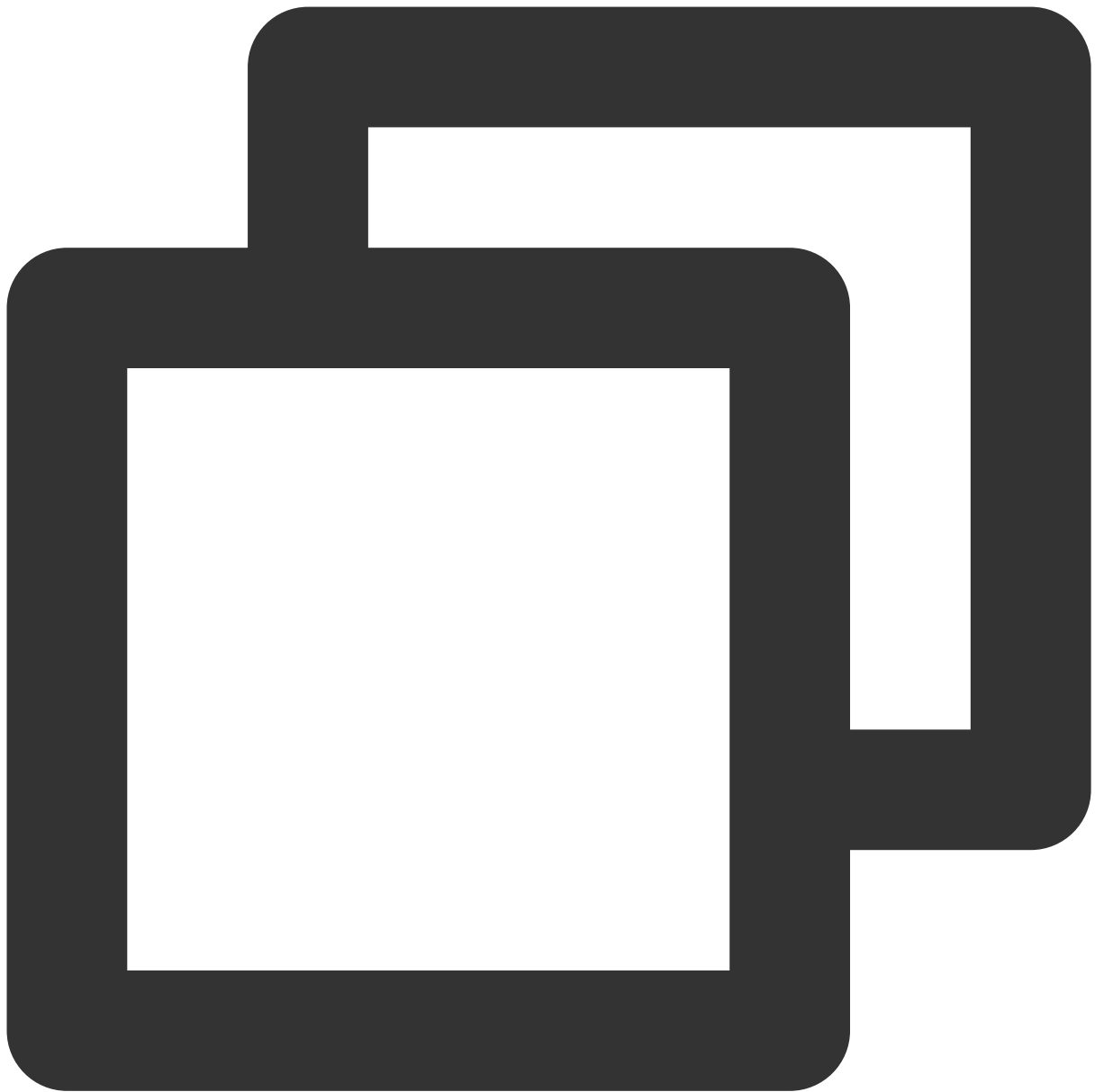
```
const _onConversationListChanged = (list) => {
  // you can use conversation list to update UI
};
```

Receiving messages

Messages can be received with the Chat SDK for React Native in two ways:

1. Listen for the persistent connection callback to get message changes and update and render the historical message list in real time.

2. Call an API to get the message history at certain time points.

Common use cases include:

1. After a new conversation is opened on the UI, request and display a certain number of historical messages.

2. Listen for the persistent connection callback to receive messages in real time and add them to the message list.

Requesting the historical message list at certain time points

To avoid affecting the pull speed, we recommend you limit the number of messages to be pulled per page to 20.

You need to dynamically record the current number of pages for the next request.

Sample code:
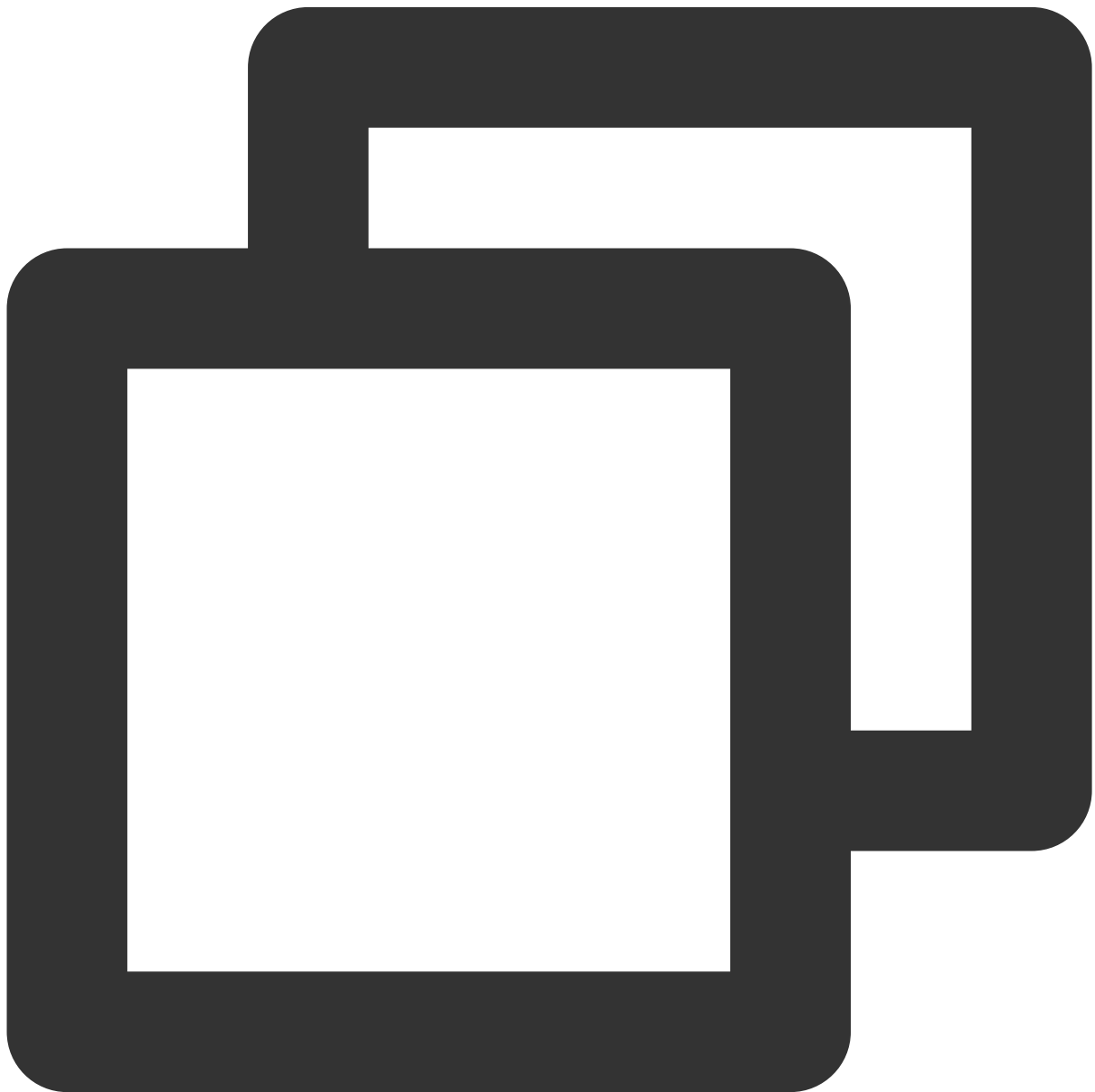
```
import { TencentImSDKPlugin } from "react-native-tim-js";
```

```
const getGroupHistoryMessageList = async () => {
  const groupID = "";
  const count = 20;
  const lastMsgID = "";
  const res = await TencentImSDKPlugin.v2TIMManager
    .getMessageManager()
    .getGroupHistoryMessageList(groupID, count, lastMsgID);
  const msgList = res.data ?? [];
  // here you can use msgList to render your message list
};
```

Listening for the persistent connection to get new messages in real time

After the historical message list is initialized, new messages are from the persistent connection

`V2TimAdvancedMsgListener.onRecvNewMessage` .

After the `onRecvNewMessage` callback is triggered, you can add new messages to the historical message list as needed.

Sample code for binding a listener:
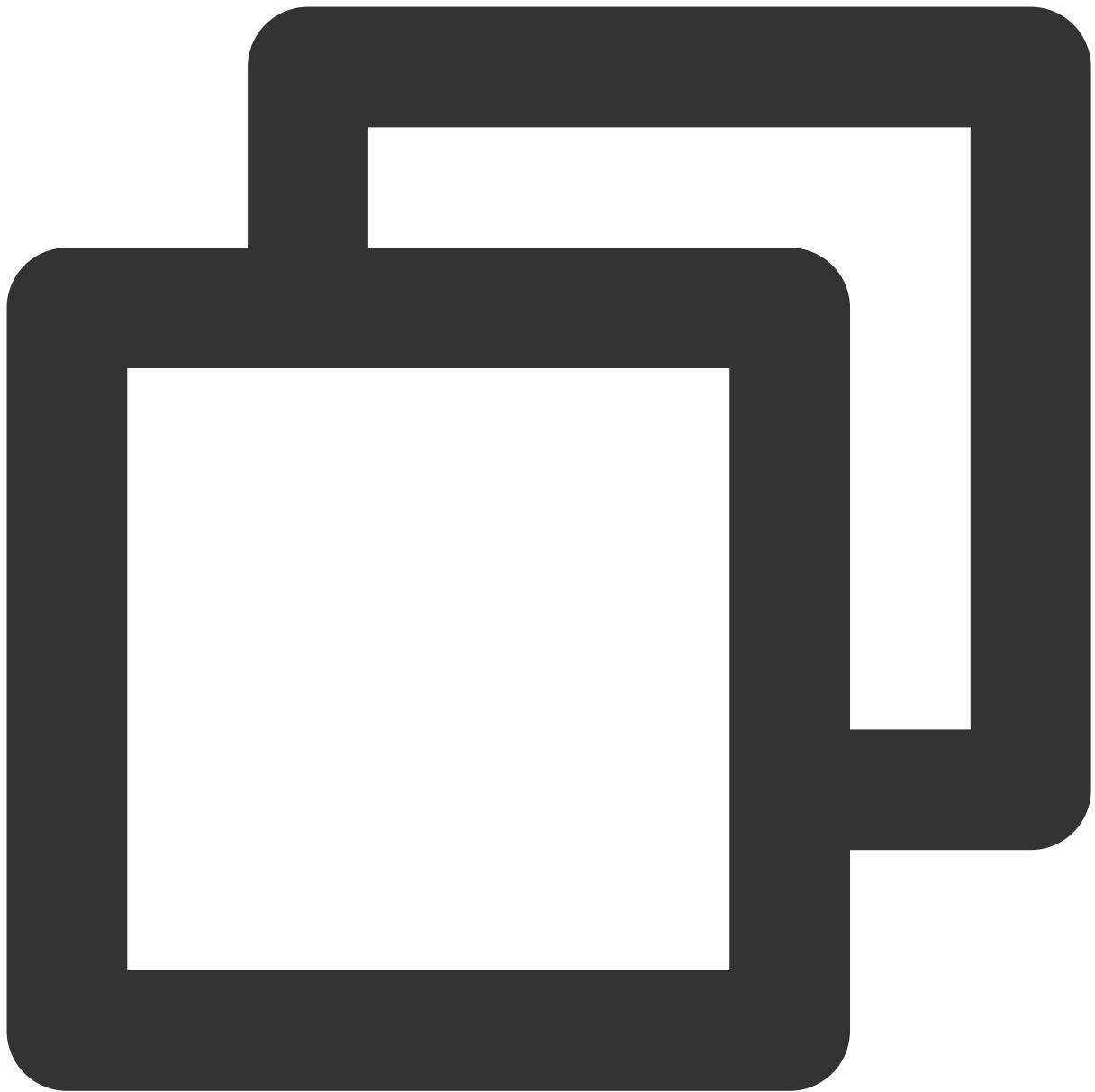
```
import { TencentImSDKPlugin } from "react-native-tim-js";

const adVancesMsgListener = {
  onRecvNewMessage: (newMsg) => {
    _onReceiveNewMsg(newMsg);
    /// ... other listeners related to message
  },
};

const addAdvancedMsgListener = () => {
  TencentImSDKPlugin.v2TIMManager
```

```
        .getMessageManager()
        .addAdvancedMsgListener(adVancesMsgListener);
    };
```

At this point, you have completed the Chat module development, and now users can send and receive messages and enter different conversations.

You can develop more features, such as group, user profile, relationship chain, offline push, and local search. For detailed directions, see here.

FAQs

What should I do if `Undefined symbols for architecture x86_64 [duplicate]` **is reported during demo running?**

See here.

Add Flutter to your existing app

Last updated：2024-04-10 16:28:29

Adding in-app chat and call modules to an Android/iOS app can be necessary as a business grows. For example, chat modules can be added to video, e-commerce, or entertainment apps to facilitate communication between users. However, adding these modules can be time-consuming and lead to an inconsistent user experience. One solution is to integrate Tencent Cloud Chat with Flutter, which allows for coding once and deploying to all platforms.

If it's not practical to rewrite the entire application in Flutter, Flutter can be integrated as a library or module. This module can then be imported into the existing Android or iOS app to render a part of the app's UI in Flutter.

This can greatly reduce your workload and allow you to quickly integrate chat and call features of the Chat SDK into your native apps.
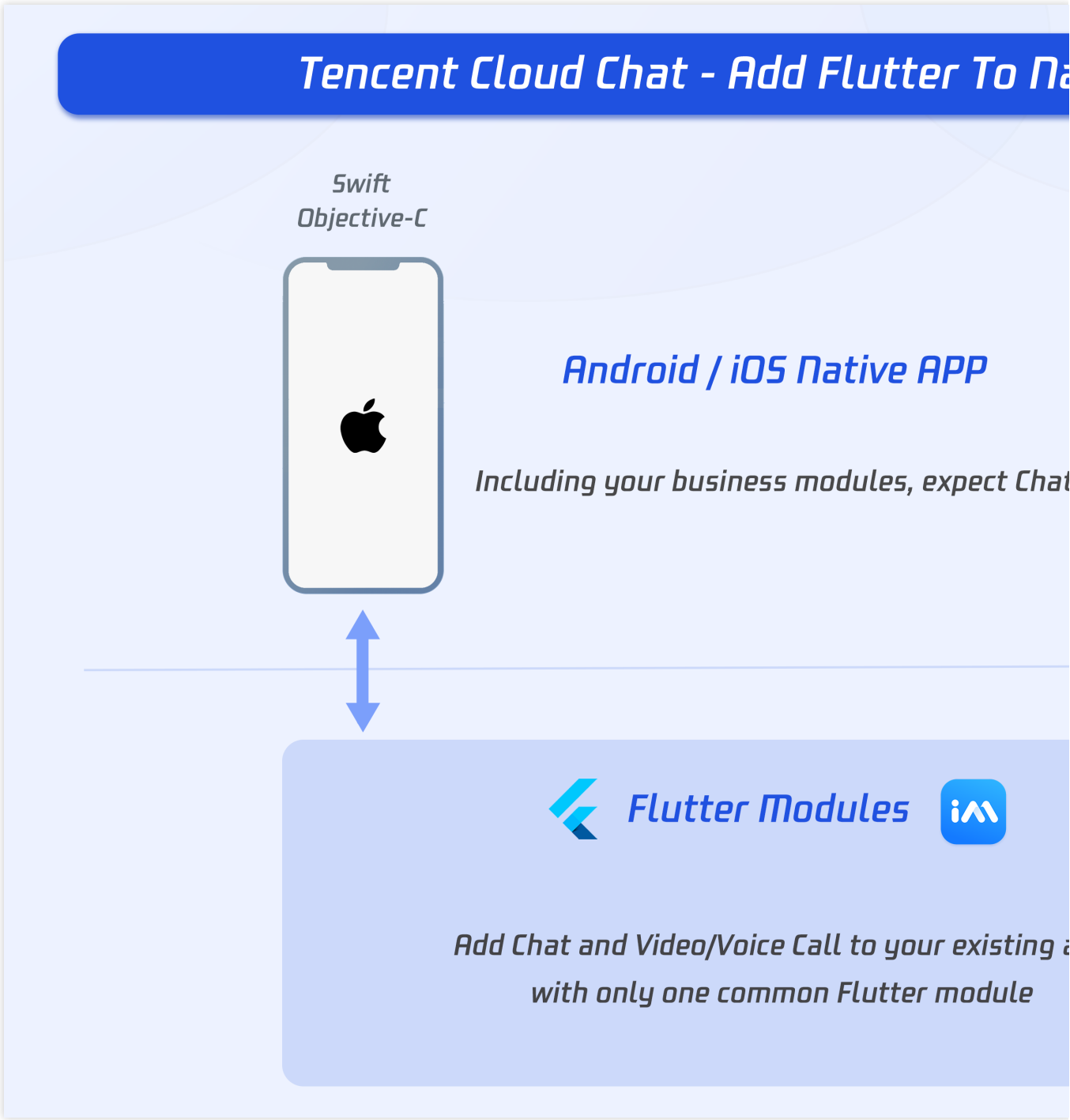
Environment requirements

Platform	Version
Flutter	Flutter 2.2.0 or later for the Chat SDK; Flutter 2.10.0 or later for the TUIKit integration component library.

Android	Android Studio 3.5 or later; devices with Android 4.1 or later for apps.
iOS	Xcode 11.0 or later. Ensure that your project has a valid developer signature.
Tencent Cloud Chat SDK	tencent_im_sdk_plugin 5.0 or later, tim_ui_kit 0.2 or later.

Sample Code：

The source code of the sample app can be found at GitHub repo.

What you need to know first

Before starting, we recommend getting to know the Tencent Cloud Chat SDK for Flutter, TUIKit, and the principles of hybrid app development with Flutter.

Tencent Cloud Chat

Overall

Before starting, you should be familiar with Tencent Cloud Chat SDK for Flutter and how it is used.

Chat offers two SDKs. One is an SDK with no UI components included. The other is UIKit, which includes UI components.

To demonstrate hybrid app development, this tutorial is written for development using TUIKit.

To learn more about the Chat SDK, see Get Started.

Modules of Tencent Cloud Chat

Tencent Cloud Chat includes two main modules: Chat and Call.

The Chat module enables you to send and receive messages, management user relationships, etc.

The Call module enables you to send and receive audio and video calls, including one-to-one call and group calls.

Adding Flutter to Native Apps

The basic principle behind hybrid app development with Flutter is to embed a Flutter module into your existing native app as a `subproject` . Because of the cross-platform nature of Flutter, you only need to develop the Flutter module once, and it can be added to both Android and iOS projects.

To launch a Flutter screen from an existing iOS/Android, you start a FlutterEngine and a FlutterViewController/FlutterActivity.

The `FlutterEngine` serves as a host to the Dart VM and your Flutter runtime, and the `FlutterViewController` / `FlutterActivity` attaches to a FlutterEngine to pass input events into Flutter and to display frames rendered by the `FlutterEngine` .

The `FlutterEngine` may have the same lifespan as your

`FlutterViewController` / `FlutterActivity` or outlive your

`FlutterViewController` / `FlutterActivity` .

Method Channel can be used to communicate between the native app and the Flutter module, for example when passing current user information, transmitting audio and video call data, or triggering offline push notifications. To trigger the method of the other end, use `invokeMethod` and to listen for the method callback from the other end using the pre-mounted `MethodCallHandler` .

Adding a Flutter Module to an Android App

Detailed documentation

This method involves adding the Flutter module as a dependency of your existing app in Gradle. There are two ways to achieve this. The first is using the AAR mechanism to create generic Android AARs as intermediaries that package your Flutter module. This is good when your downstream app builders don't want to have the Flutter SDK installed. But, it adds one more build step if you build frequently.

The other is using the source code subproject mechanism, which is a convenient one-click build process, but requires the Flutter SDK. This is the mechanism used by the Android Studio IDE plugin.

Option A - Depend on the Android Archive (AAR)

This option packages your Flutter library as a generic local Maven repository composed of AARs and POMs artifacts. This option allows your team to build the host app without installing the Flutter SDK. You can then distribute the artifacts from a local or remote repository.

It's recommended to use this option for the release version of your app.

Steps:

Run the following command on your Flutter module.
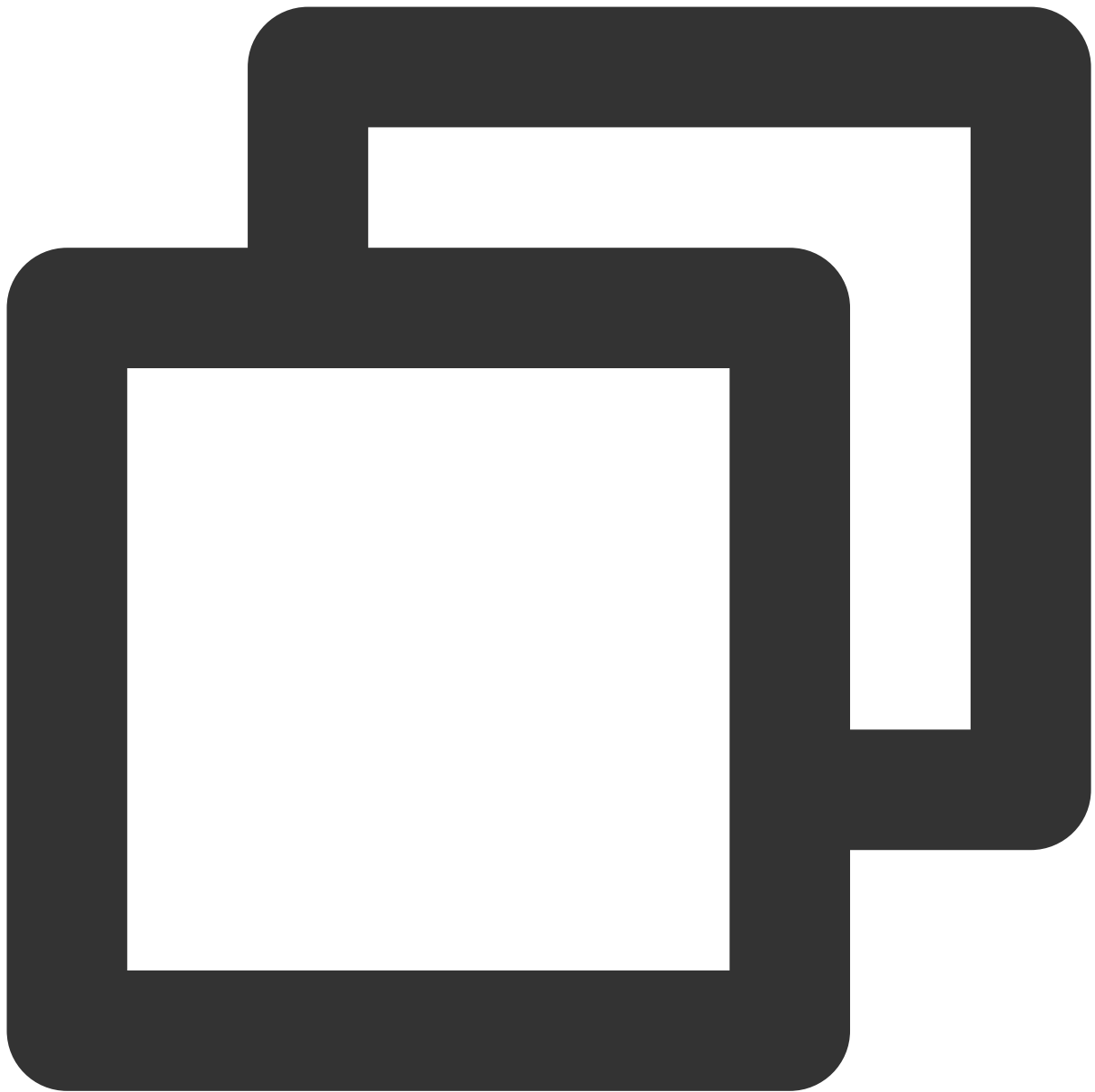
```
flutter build aar
```

Then, follow the on-screen instructions to integrate.

Your app now includes the Flutter module as a dependency.

Option B - Depend on the module's source code

This option enables a one-step build for both your Android project and Flutter project.

This option is convenient when you work on both parts simultaneously and rapidly iterate, but your team must install the Flutter SDK to build the host app.

It's recommended to use this option when development and debugging.

Steps:

Include the Flutter module as a subproject in the host app's `settings.gradle` :
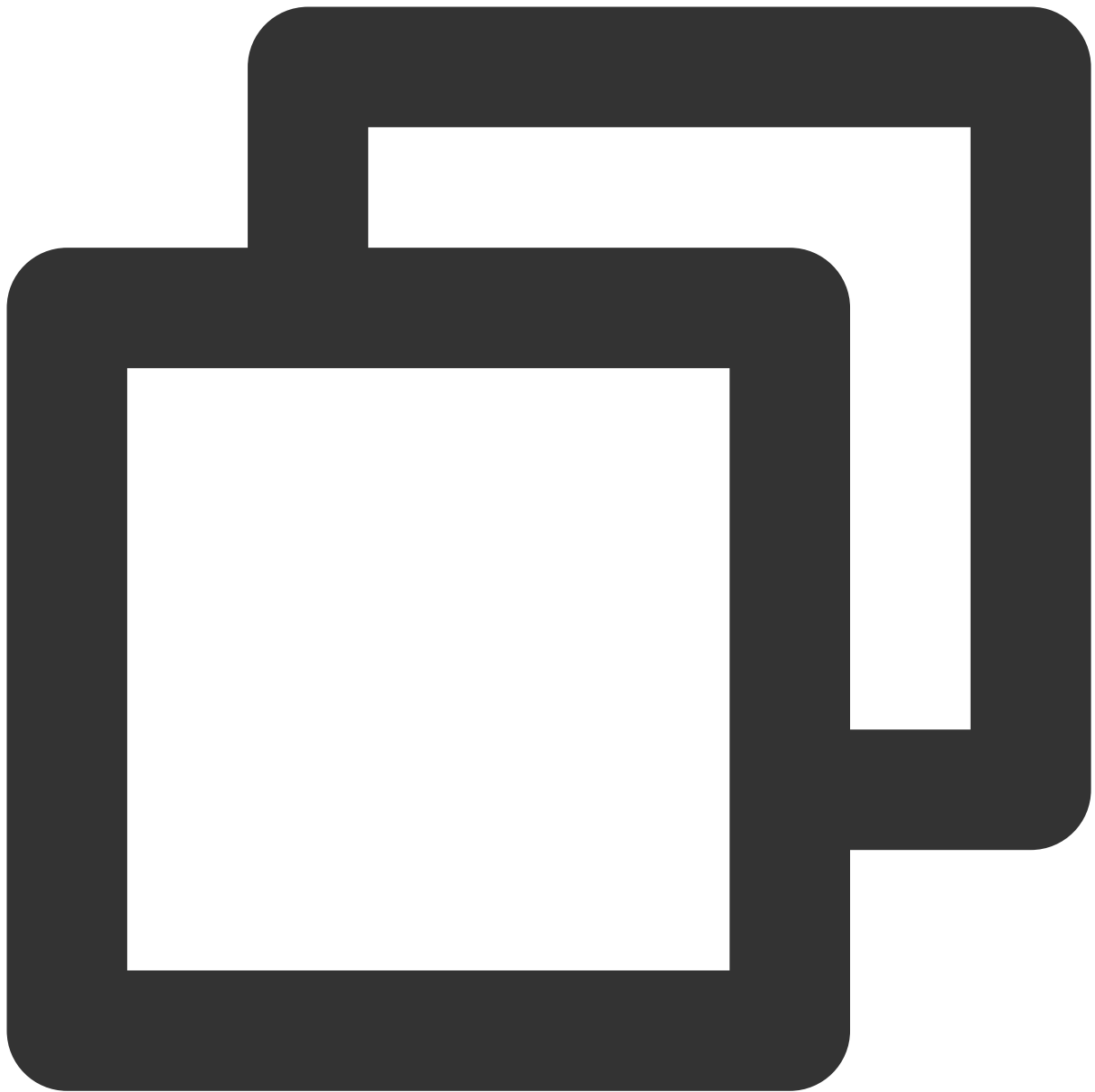
```
// Include the host app project.
include ':app'                                    // assumed existing content

// Add the following lines to your project
setBinding(new Binding([gradle: this]))
evaluate(new File(
settingsDir.parentFile,
'tencent_chat_module/.android/include_flutter.groovy'
))
```

Introduce an `implementation` dependency on the Flutter module from your app:
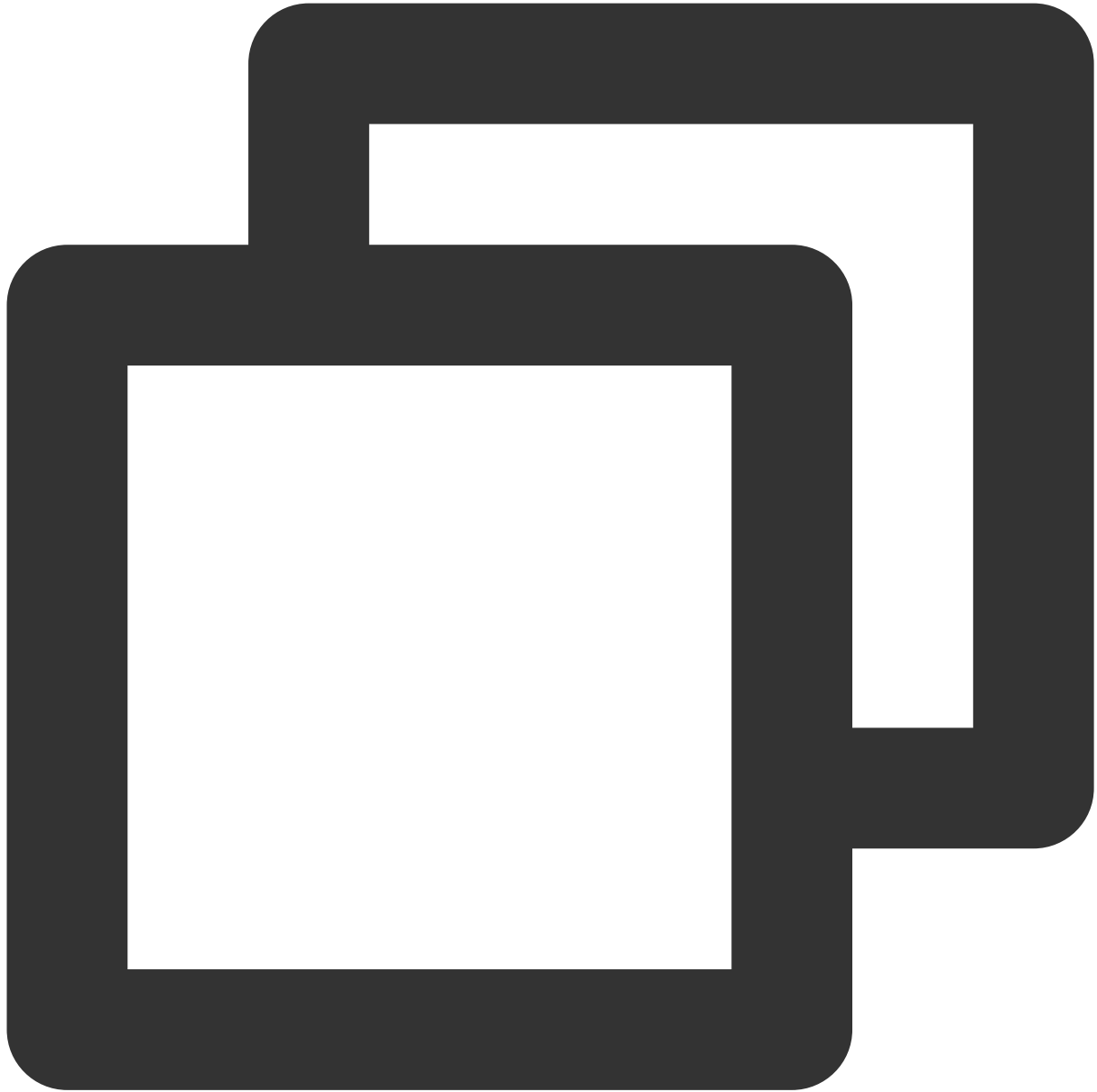
```
dependencies {
implementation project(':flutter')
}
```

Your app now includes the Flutter module as a dependency.

Adding a Flutter Module to an iOS App

[Detailed documentation](#)

There are two ways to embed Flutter in your existing application.

Use the CocoaPods dependency manager and install Flutter SDK. (Recommended)

Create frameworks for the Flutter engine, your compiled Dart code, and all Flutter plugins. Manually embed the frameworks, and update your existing application's build settings in Xcode.

Note:

Your app does not run on a simulator in Release mode because Flutter does not yet support outputting x86/x86_64 ahead-of-time (AOT) binaries for your Dart code. You can run in Debug mode on a simulator or a real device, and Release on a real device.

To run your app on a simulator follow the instructions in the bottom of section embed the frameworks.

Option A - Embed with CocoaPods and the Flutter SDK

This method requires every developer working on your project to have a locally installed version of the Flutter SDK. Simply build your application in Xcode to automatically run the script to embed your Dart and plugin code.

This allows rapid iteration with the most up-to-date version of your Flutter module without running additional commands outside of Xcode.

It's recommended to use this option for development and debugging.

Steps:

Add the following lines to your `Podfile` :
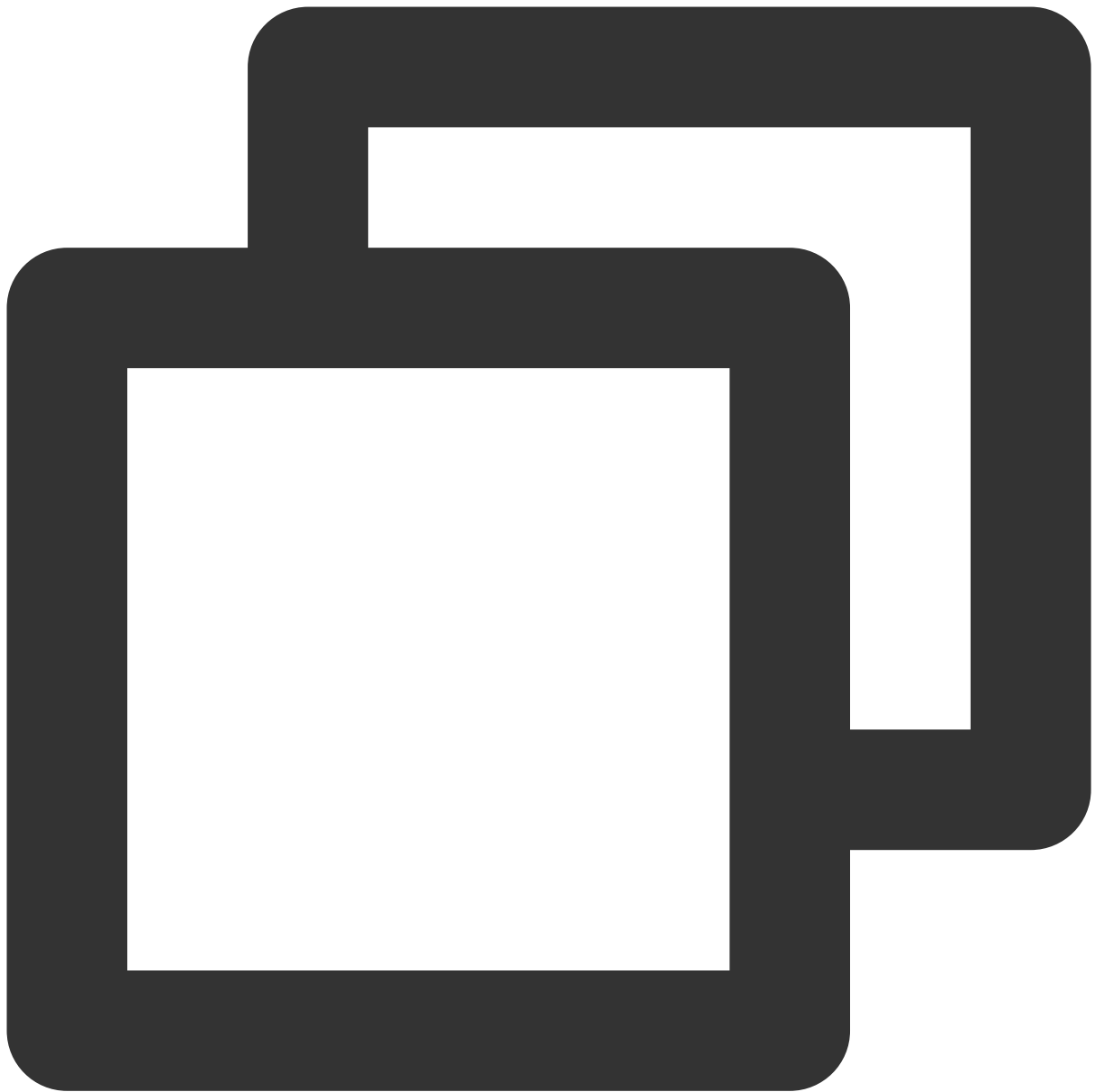
```
// The path of your Flutter module
flutter_chat_application_path = '../tencent_chat_module'

load File.join(flutter_chat_application_path, '.ios', 'Flutter', 'podhelper.rb')
```

For each Podfile target that needs to embed Flutter, call

`install_all_flutter_pods(flutter_application_path)` .

Tencent Cloud
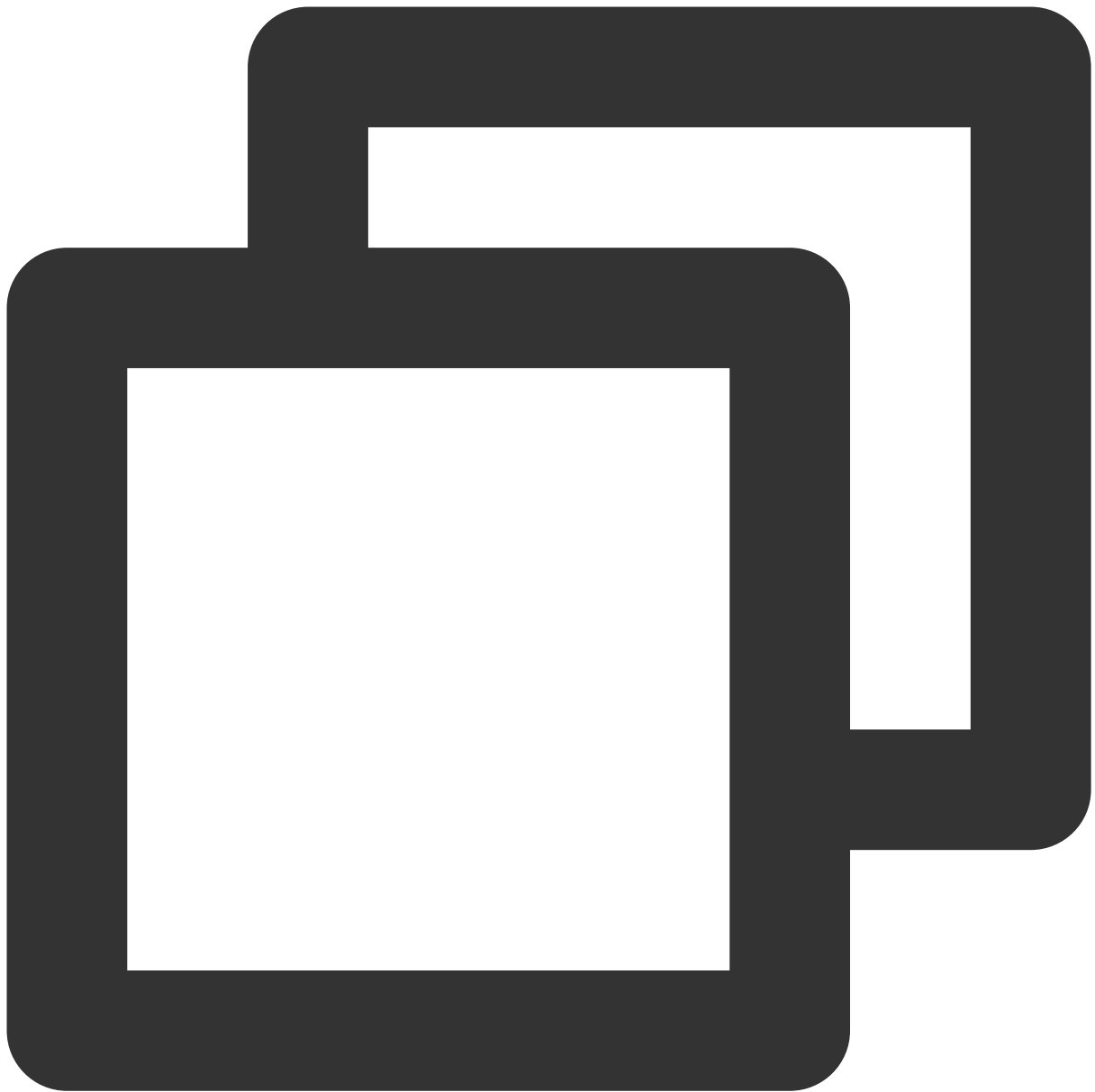
```
target 'MyApp' do
install_all_flutter_pods(flutter_chat_application_path)
end
```

In the `Podfile` 's `post_install` block, call `flutter_post_install(installer)` , and with the statement of necessary permissions.
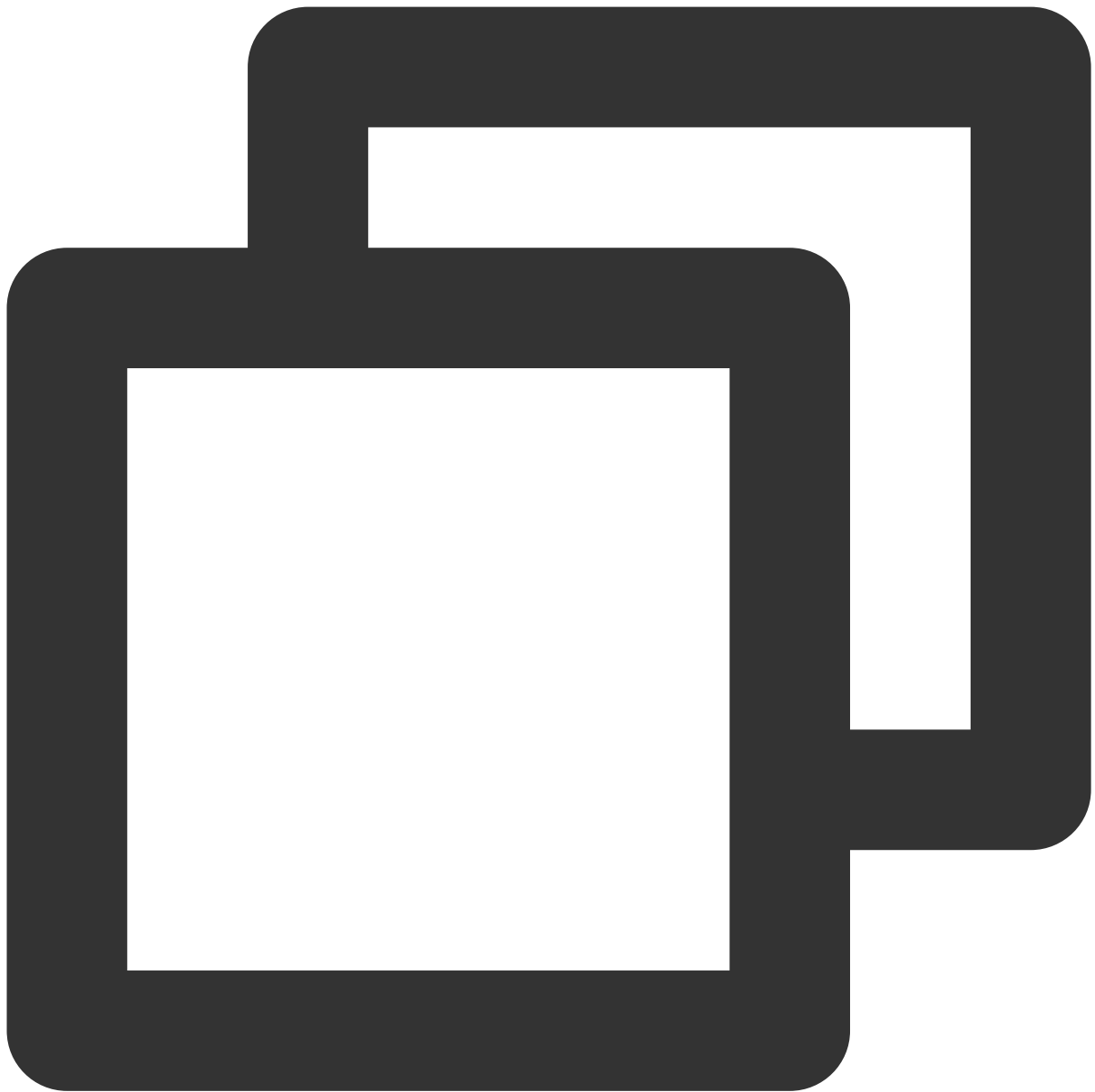
```
post_install do |installer|
flutter_post_install(installer) if defined?(flutter_post_install)
installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
        config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
          '$(inherited)',
          'PERMISSION_MICROPHONE=1',
          'PERMISSION_CAMERA=1',
          'PERMISSION_PHOTOS=1',
        ]
```

```
        end
   end
  end
```

Run `pod install` .

Note:

When you change the Flutter plugin dependencies in `tencent_chat_module/pubspec.yaml` , run `flutter pub get` in your Flutter module directory to refresh the list of plugins read by the `podhelper.rb` script. Then, run pod install again from the root directory of your application.

You may need to run `arch -x86_64 pod install --repo-update` on the Mac with Apple Silicon, like M1 or M2.

The `podhelper.rb` script embeds your plugins, `Flutter.framework` , and `App.framework` into your project.

Option B - Embed frameworks in Xcode

Alternatively, you can generate the necessary frameworks and embed them in your application by manually editing your existing Xcode project.

You may do this if members of your team can't locally install Flutter SDK and CocoaPods, or if you don't want to use CocoaPods as a dependency manager in your existing applications.

You must run `flutter build ios-framework` every time you make code changes in your Flutter module.

It's recommended to use this option for the released version.

Steps:

Run the following command on your Flutter module.

The following example assumes that you want to generate the frameworks to `some/path/MyApp/Flutter/` .

Tencent Cloud
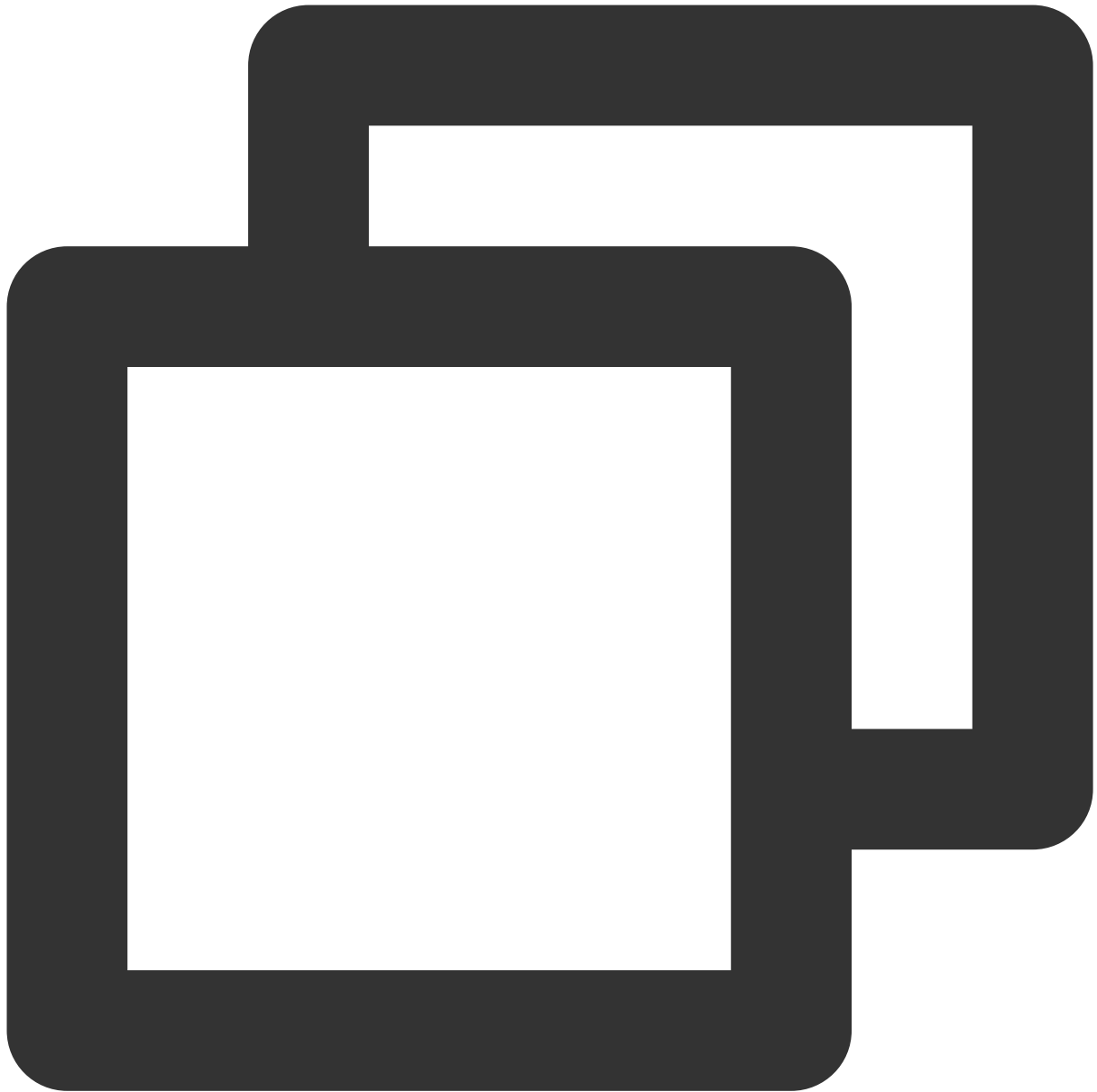

```
flutter build ios-framework --output=some/path/MyApp/Flutter/
```

Embed and link the generated frameworks into your existing application in Xcode.

Adding Flutter Engines

It's recommended to add the Flutter module to your existing application.

You need to add the Chat and Call modules to your native app using Flutter engines. For details, see here.

There are two options for creating and managing Flutter engines: create a single `FlutterEngine` or create two `FlutterEngines` with `FlutterEngineGroup` .

Mode	Introduction	Pros	Cons	Sample Code
Single FlutterEngine	Both Chat and Call integrate into one `FlutterEngine`	Convenient	Since the Call module needs to automatically display the incoming call page when a call is received, it needs to be forced to redirect to the Flutter page, resulting in a poorer experience.	GitHub
Multiple FlutterEngines	The Chat and Call modules are located in two separate Flutter engines	The Call module exists independently in a Flutter engine, with independent page control. When a call comes in, the Flutter page is shown, which does not affect the page where the user is currently located, providing a better experience.	Minimize for the calling page is not allowed.	GitHub

In addition to the methods above, you can also integrate a native Chat SDK and Flutter SDK. For details, see here, and sample code can be found from GitHub.

Solution A: Multiple FlutterEngines (Recommended)

The advantage of using multiple Flutter instances is that each instance is independent and maintains its own internal navigation stack, UI, and application states. This simplifies the overall application code's responsibility for state keeping and improves modularity. More details on the scenarios motivating the usage of multiple Flutters can be found at docs.flutter.dev/go/multiple-flutters.
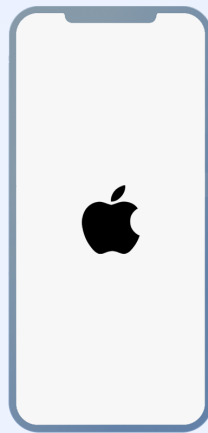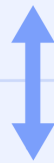
The primary API for adding multiple Flutter instances on both Android and iOS is based on a new `FlutterEngineGroup` class to construct `FlutterEngine` s, rather than the `FlutterEngine` constructors used in the Solution B: Single FlutterEngine.

Whereas the `FlutterEngine` API was direct and easier to consume, the `FlutterEngine` spawned from the same `FlutterEngineGroup` have the performance advantage of sharing many of the common, reusable resources such as the GPU context, font metrics, and isolate group snapshot, leading to a faster initial rendering latency and lower memory footprint.

In our single `FlutterEngineGroup` is used to manage the two `FlutterEngine` s, including Chat and Call modules.

You can refer to the sample code from [GitHub repo](#) this module.

Developing the Flutter Module

To embed Flutter into your existing application, you must first create a Flutter module.

From the command line, run:

```
cd some/path/
flutter create --template module tencent_chat_module
```

Tencent Cloud

A Flutter module project is created at `some/path/tencent_chat_module/` . From that directory, you can run the same flutter commands you would in any other Flutter project, like `flutter run --debug` or `flutter build ios` . You can also run the module in Android Studio/IntelliJ or VS Code with the Flutter and Dart plugins. This project contains a single-view example version of your module before it's embedded in your existing application, which is useful for incrementally testing the Flutter-only parts of your code.

The `tencent_chat_module` module directory structure is similar to a normal Flutter application:

```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
```

```
|       └── Flutter/podhelper.rb
├── lib/
|       └── main.dart
├── test/
└── pubspec.yaml
```

Now, you can add code within `lib/` .

The structure of `lib/`

Note:

The following structure and code are for demonstration purposes only. You can modify them to meet your actual

needs.

Within `lib/` , you need to create three directories: call, chat, and common. These directories are used for the Call

module, Chat module, and some common classes, respectively.

```
tencent_chat_module/
├── lib/
│   └── call/
│   └── chat/
│   └── common/
```

Common model classes

Create a new file named `common/common_model.dart` , as shown below. Create two classes in this file to define the communication standard between Flutter and the native application.
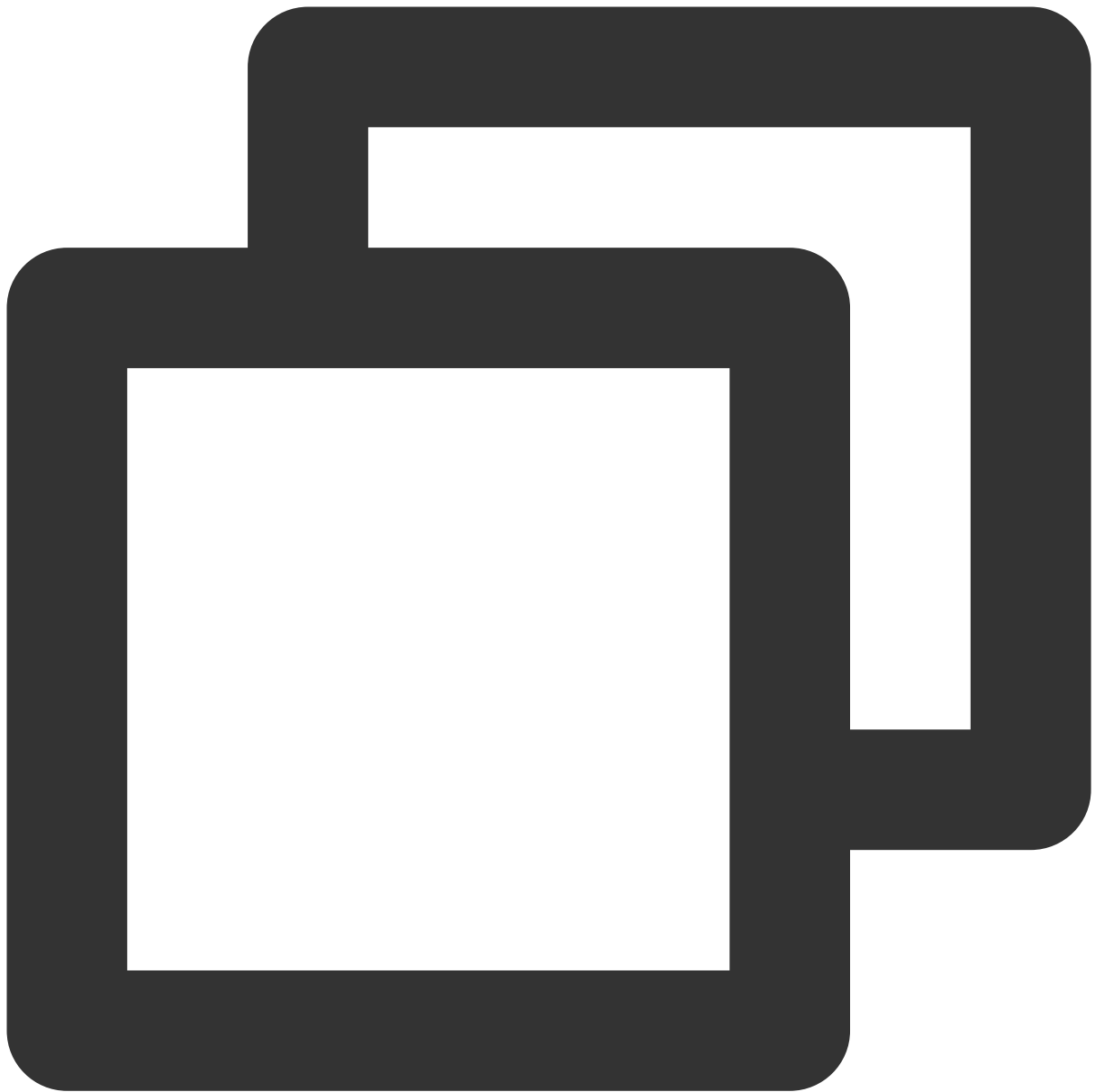
```
class ChatInfo {
String? sdkappid;
String? userSig;
String? userID;

ChatInfo.fromJSON(Map<String, dynamic> json) {
  sdkappid = json["sdkappid"].toString();
  userSig = json["userSig"].toString();
  userID = json["userID"].toString();
}
```

```
  Map<String, String> toMap(){
    final Map<String, String> map = {};
    if(sdkappid != null){
      map["sdkappid"] = sdkappid!;
    }
    if(userSig != null){
      map["userSig"] = userSig!;
    }
    if(userID != null){
      map["userID"] = userID!;
    }
    return map;
  }
}

class CallInfo{
String? userID;
String? groupID;

CallInfo();

CallInfo.fromJSON(Map<String, dynamic> json) {
  groupID = json["groupID"].toString();
  userID = json["userID"].toString();
}

Map<String, String> toMap(){
  final Map<String, String> map = {};
  if(userID != null){
    map["userID"] = userID!;
  }
  if(groupID != null){
    map["groupID"] = groupID!;
  }
  return map;
}
}
```

Chat Module

The following files and code are located in the `lib/chat` directory.

1. Create a file, model.dart, used as a state container.

This model is used to initialize and maintain the instance of Tencent Cloud Chat, offline line push module, global state,

and the communication with native apps.

 It's the core of the Chat module.

For detailed implementation, refer to the source code of the sample app, paying attention to the following three functions:

Future_handleMessage(MethodCall call): Listens for events from native app, including login and notification click events.

Future handleClickNotification(Map< String, dynamic> msg): The function invoked by the callback after clicking the notification.

Future initChat(): Initialize and log in Tencent Cloud Chat SDK and offline push plugin, upload token. This method uses the sync lock mechanism to ensure that only one can be executed at the same time, and after the initialization is successful, it will not be executed repeatedly.

Note:

Please configure the offline push before uploading the token and use this capability, see Integrating Offline Push.

2. Create a file, `chat_main.dart` . This is also used as the home page of the chat module.

Also, used as the home page of the chat module.

It shows the loading status before logged in, followed by the conversation list.

Besides, the current status of the application needs to be reported to the Tencent Cloud Chat backend upon each foreground/background switch from here.

Detailed implementation can refer to the sample code from GitHub repo.

2.1 Create a file, `push.dart` , used for maintaining the offline push plugin. Detailed implementation can refer to the sample code from GitHub repo.

2.2 Create a file, `conversation.dart` , used for implementing conversation list widget `TIMUIKitConversation` . Detailed implementation can refer to the sample code from GitHub repo.

2.3 Create a file, `user_profile.dart` , used to implement the user profile widget `TIMUIKitProfile` . Detailed implementation can refer to the sample code from GitHub repo.

2.4 Create a file, `group_profile.dart` , used to implement group profile widget `TIMUIKitGroupProfile` . Detailed implementation can refer to the sample code from GitHub repo.

2.5 Create a file, `chat.dart` , used for implementing the history message list and sending messages widget `TIMUIKitChat` . This page can also navigate to `user_profile.dart` and `conversation.dart` . Detailed implementation can refer to the sample code from GitHub repo.

At this point, the Chat module has been developed with the following structure:

```
tencent_chat_module/
├── lib/
│   └── call/
│       └── chat.dart
│       └── model.dart
│       └── chat_main.dart
│       └── push.dart
│       └── conversation.dart
│       └── user_profile.dart
│       └── group_profile.dart
│   └── chat/
```

```
|    └── common/
```

Call Module

This module is used for voice call and video call, provided by our calling plugin.

The key function of this module is to automatically show the call page by calling the native method when a new call invitation is received, as well as to accept the call request forwarded by the Chat module and actively initiate the call.

The following files and code are located in the `lib/call` directory.

1. Create a file, `model.dart`, used as a state container.

This model is used for initializing and maintaining the instance of Calling plug-in, global state, and the communication with native apps.

Is the core of the Call module.

Detailed implementation can refer to the source code of the sample app, while it's recommended to focus on these two functions:

_onRtcListener = TUICallingListener(...): The listener of the calling events, notify native to show this page, when receiving a new call.

Future_handleMessage(MethodCall call): The listener of the method channel call events, mainly used for initiating a call to others, when receiving the request from the Chat module, via native.

2. Create a file, `call_main.dart`, used as the main entry point of the Call module.

The `navigatorKey` used for launching the call page should be added here.

Detailed implementation can refer to the sample code from GitHub repo.

Configure the entry point for each module

After completing the three parts above, you can configure the entry point for each module, which serve as the entry for the Flutter engine.

1. Default entry

Open `lib/main.dart`, modify the default main functions to return an empty `MaterialApp`.

This method serves as the default entry point for the Flutter module. Under the management of FlutterEngineGroup in a multi-engine Flutter engine scenario, if no child Flutter engine sets any entry point, this method will not be used.

For example, in this tutorial, this default main() method is not used.

Tencent Cloud
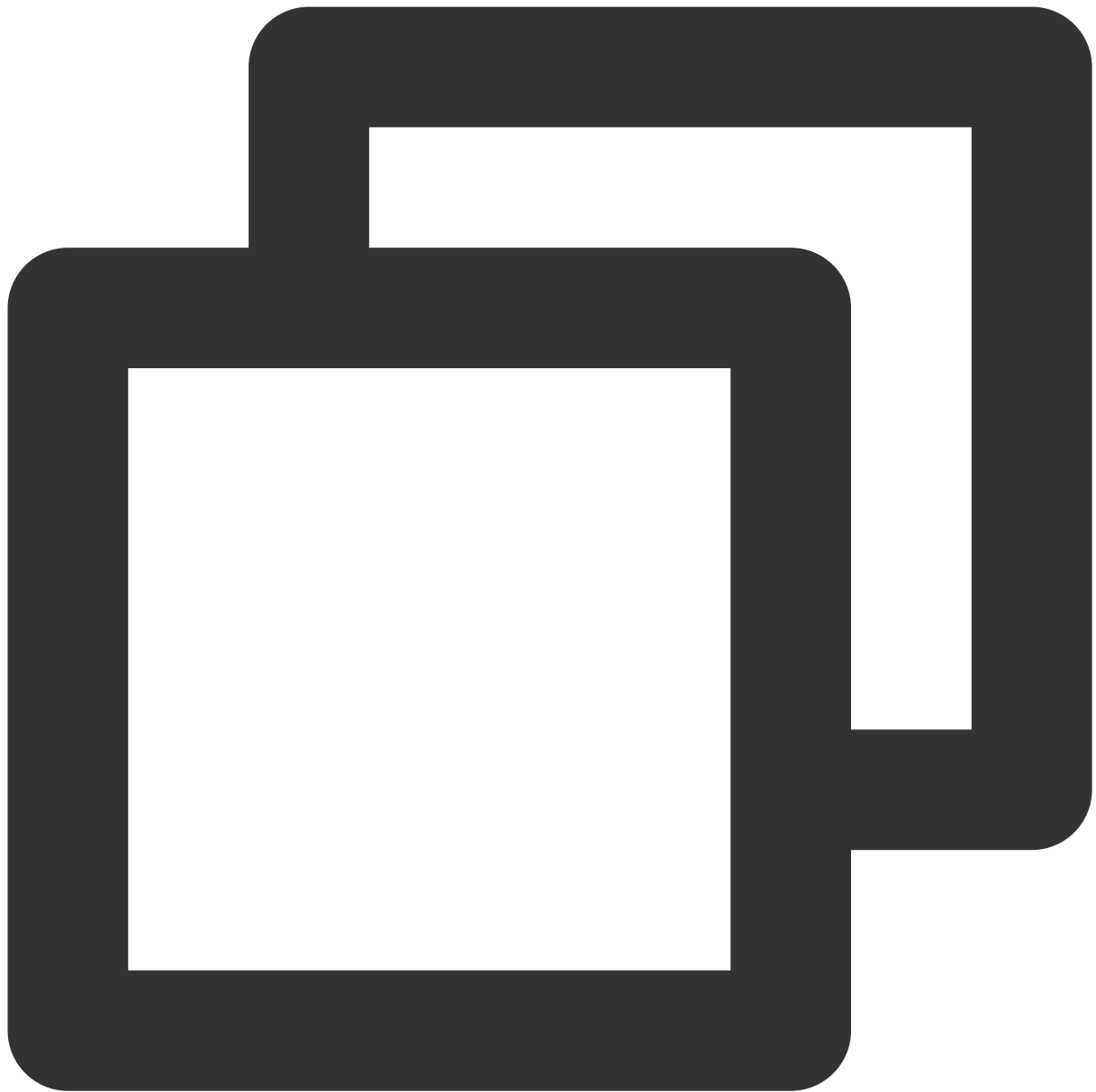
```
void main() {
WidgetsFlutterBinding.ensureInitialized();

runApp(MaterialApp(
  title: 'Flutter Demo',
  theme: ThemeData(
    primarySwatch: Colors.blue,
  ),
  home: Container(),
));
}
```

2. The entry for Chat module

Use `@pragma('vm:entry-point')` to mark a method as an entry point. The method named `chatMain` is the name of the entry.

In the native code, this name is also used to create the corresponding `FlutterEngine`.

Use the global `ChangeNotifierProvider` for state management to maintain `ChatInfoModel` data and business logic.
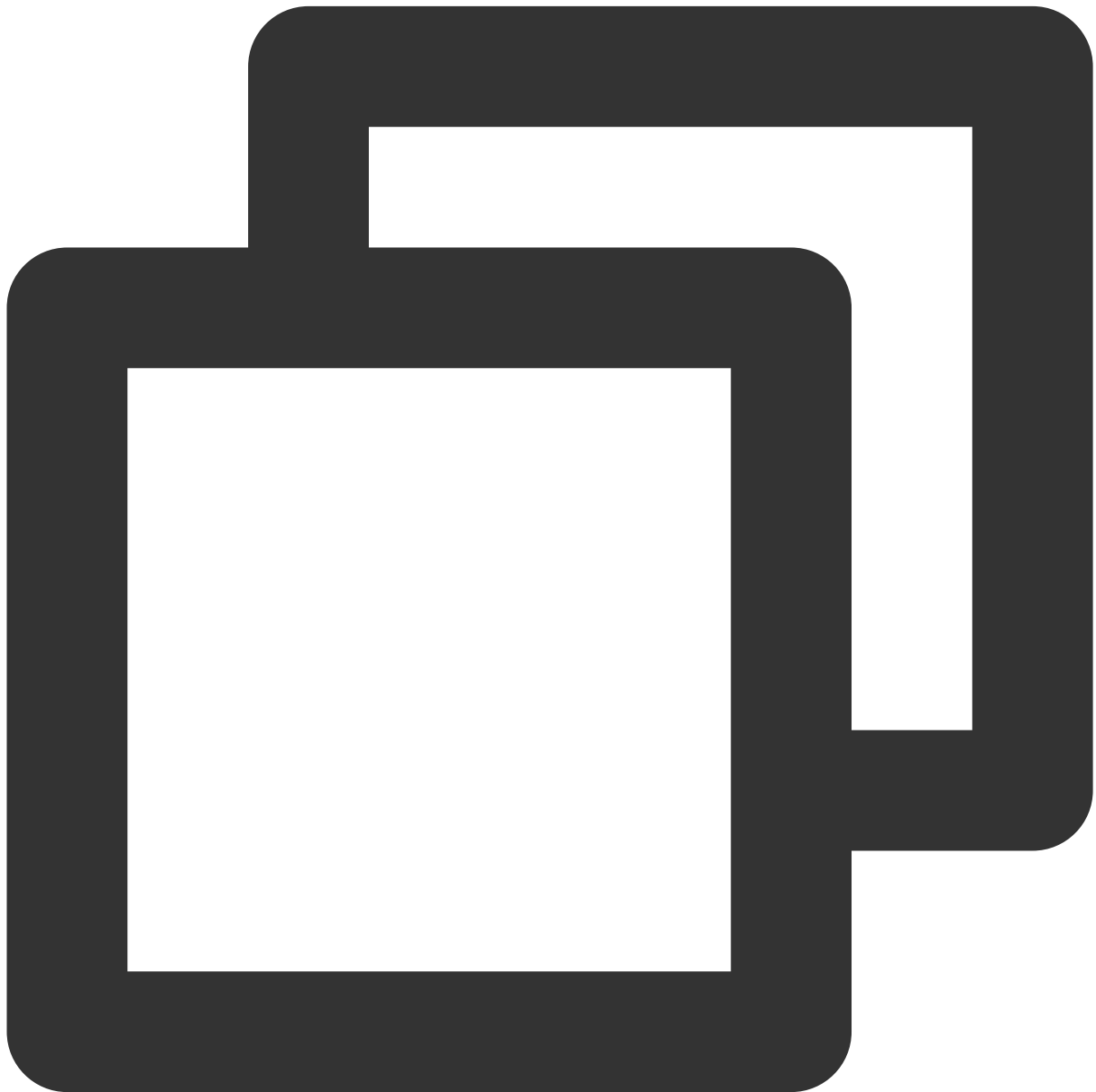

```
@pragma('vm:entry-point')
void chatMain() {
```

```
// This call ensures the Flutter binding has been set up before creating the
// MethodChannel-based model.
WidgetsFlutterBinding.ensureInitialized();

final model = ChatInfoModel();

runApp(
  ChangeNotifierProvider.value(
    value: model,
    child: const ChatAPP(),
  ),
);
}
```

3. The entry for Call module

This entry point is named as `callMain` .

Use global `ChangeNotifierProvider` status management to maintain `CallInfoModel` data and business logic.
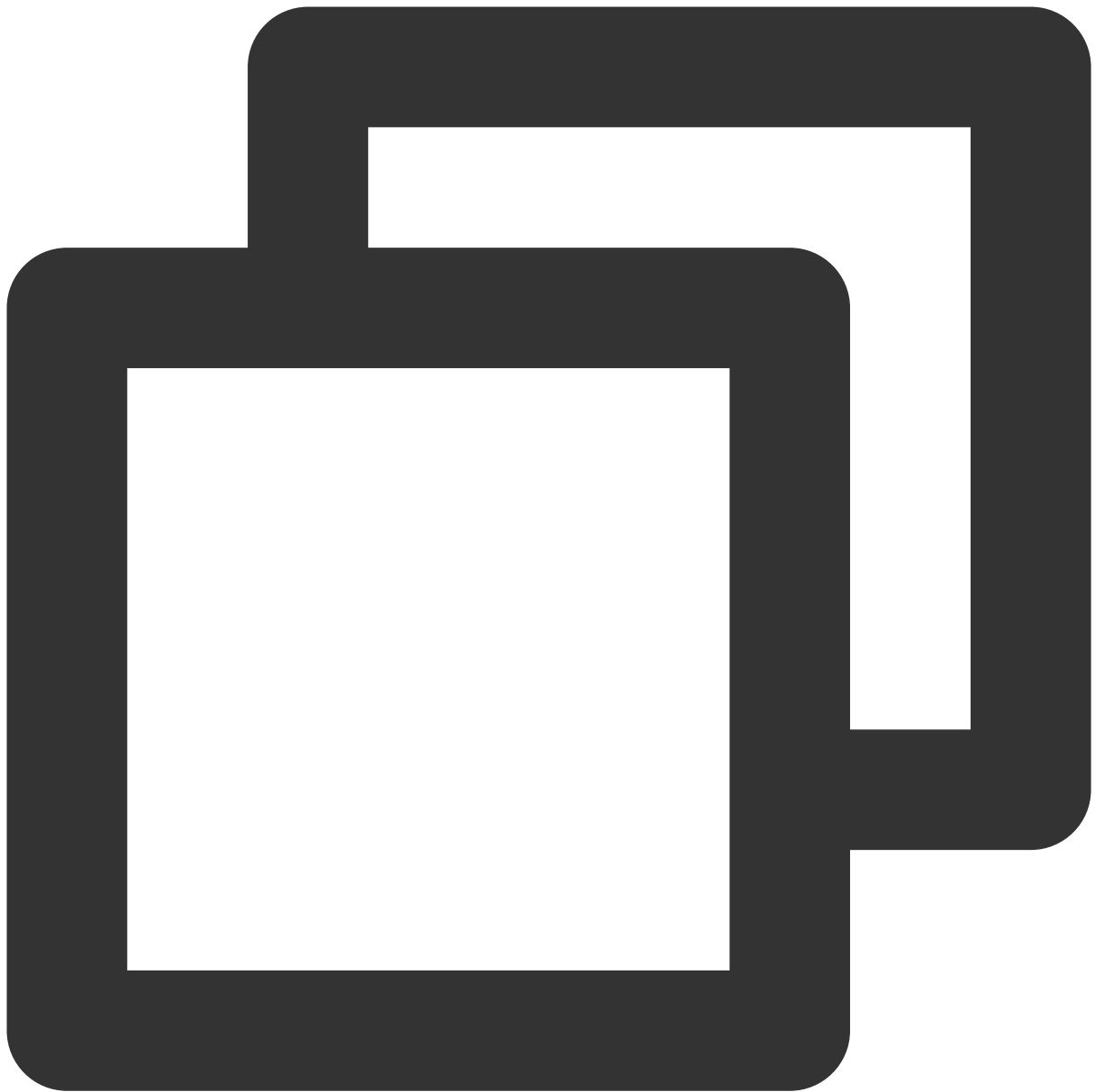
```
@pragma('vm:entry-point')
void callMain() {
// This call ensures the Flutter binding has been set up before creating the
// MethodChannel-based model.
WidgetsFlutterBinding.ensureInitialized();

final model = CallInfoModel();

runApp(
  ChangeNotifierProvider.value(
    value: model,
```

```
      child: const CallAPP(),
    ),
  );
  }
```

At this point, the Dart code for the Flutter Module has been completed.

Now, let's take a look at the native integration for your existing app.

iOS Native development

In this section, `Swift` is used as an example, but `Objective-C` is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your iOS project within XCode.

If your existing application (MyApp) doesn't already have a `Podfile`, follow the CocoaPods getting started guide to add a Podfile to your project.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing iOS app.

FlutterEngineGroup

Create a `FlutterEngineGroup` **to maintain and manage the** `FlutterEngine` **s.**

The proper place to create a `FlutterEngineGroup` is specific to your host app. As an example, we demonstrate

creating a `FlutterEngineGroup` , exposed as a property, on app startup in the app delegate.

Add the following to `AppDelegate.swift` .
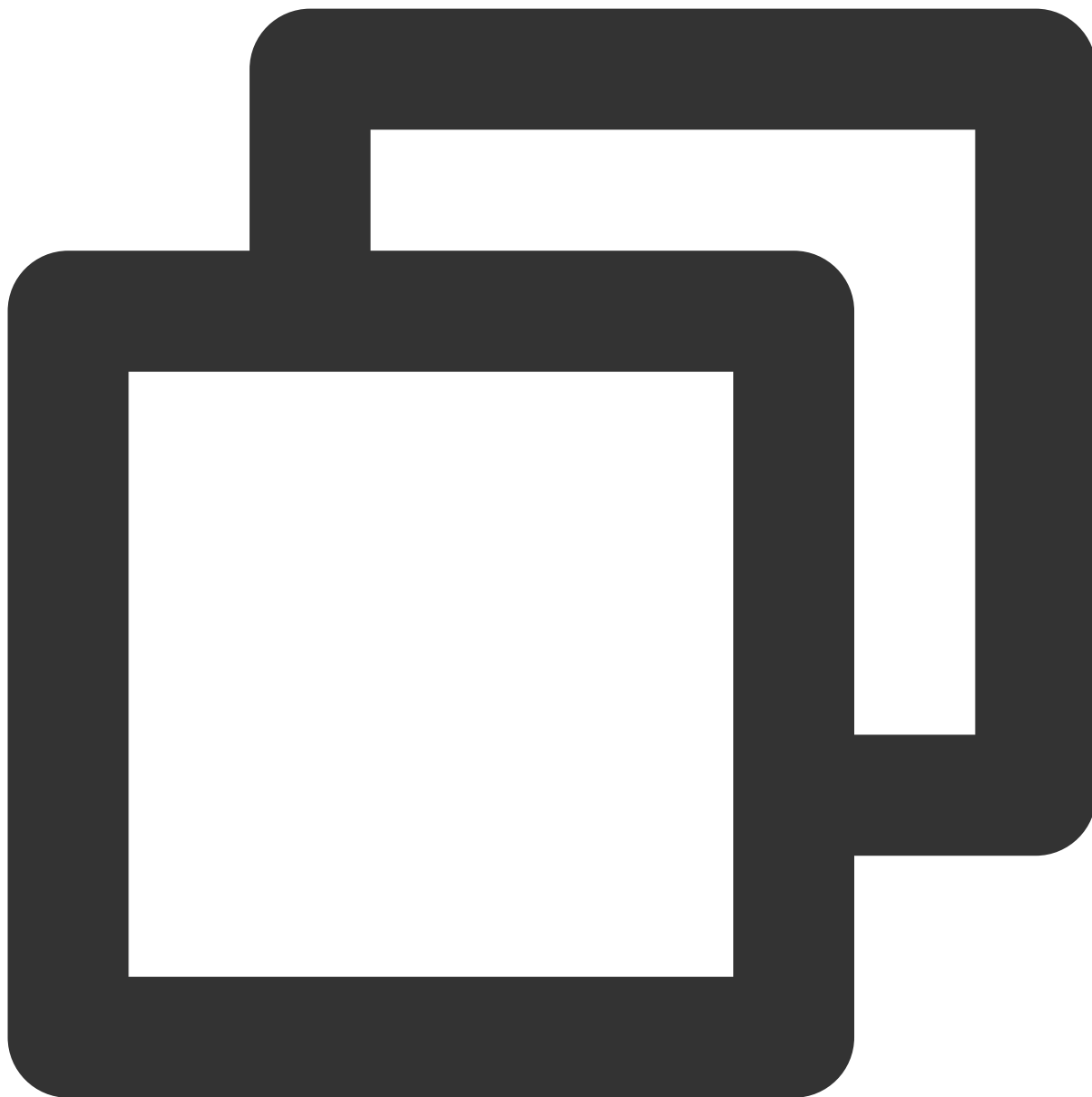
```
@UIApplicationMain
class AppDelegate: FlutterAppDelegate {
lazy var flutterEngines = FlutterEngineGroup(name: "chat.flutter.tencent", project:
...
}
```

Create a singleton static object to hold `FlutterEngine` **s.**

This singleton is used for managing those `FlutterEngine` s in one place, and provides methods to the whole

project to invoke methods related to the Flutter module.

In the sample code, a new navigator is used for the Chat ViewController. The Call ViewController is maintained dynamically with present and dismiss.

Create a new file named FlutterUtils.swift. For detailed code, refer to the sample code.

Mainly focus on:

private override init(): Initialize each Flutter instance, register method channel events.

func reportChatInfo(): Report the current user info to the Flutter module, for initialization and login Tencent Cloud Chat SDK.

func launchCallFunc(): Present the ViewController for Call module, invoked when new call income or user active it manually from Chat module.

func triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

Listen for and forward the notification click event

The initialization/token reporting/click event corresponding to the offline push notification is handled in the Flutter Chat module. Therefore, on the native side, only the ext of the click notification event needs to be passed through.

The reason we need to do this is because the clicking event has been consumed on the native side, so it is impossible for the Flutter Push plug-in to receive this event.

Add the following code to `AppDelegate.swift` .

```swift
20
21  @UIApplicationMain
22  class AppDelegate: FlutterAppDelegate {
23    lazy var flutterEngines = FlutterEngineGroup(name: "chat.flutter.tencent", project: nil)
24
25    override func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
26
27      if #available(iOS 10.0, *) {
28        UNUserNotificationCenter.current().delegate = self
29      }
30
31      if let remoteNotification = launchOptions?[UIApplication.LaunchOptionsKey.remoteNotification]{
32        let notificationExt: [AnyHashable:Any] = remoteNotification as! [AnyHashable:Any]
33        let remoteNotificationString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
34        FlutterUtils.shared.triggerNotification(msg: remoteNotificationString)
35      }
36
37      return super.application(application, didFinishLaunchingWithOptions: launchOptions);
38    }
39
40    override func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, withCompletionHandler completio
        @escaping () -> Void) {
41      let notificationExt: Dictionary = response.notification.request.content.userInfo
42      let notificationExtString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
43      FlutterUtils.shared.triggerNotification(msg: notificationExtString)
44    }
45
46  }
```

At this point, the implementation for iOS is complete.

Android Native Development

Here, Kotlin is used as an example, but Java is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.
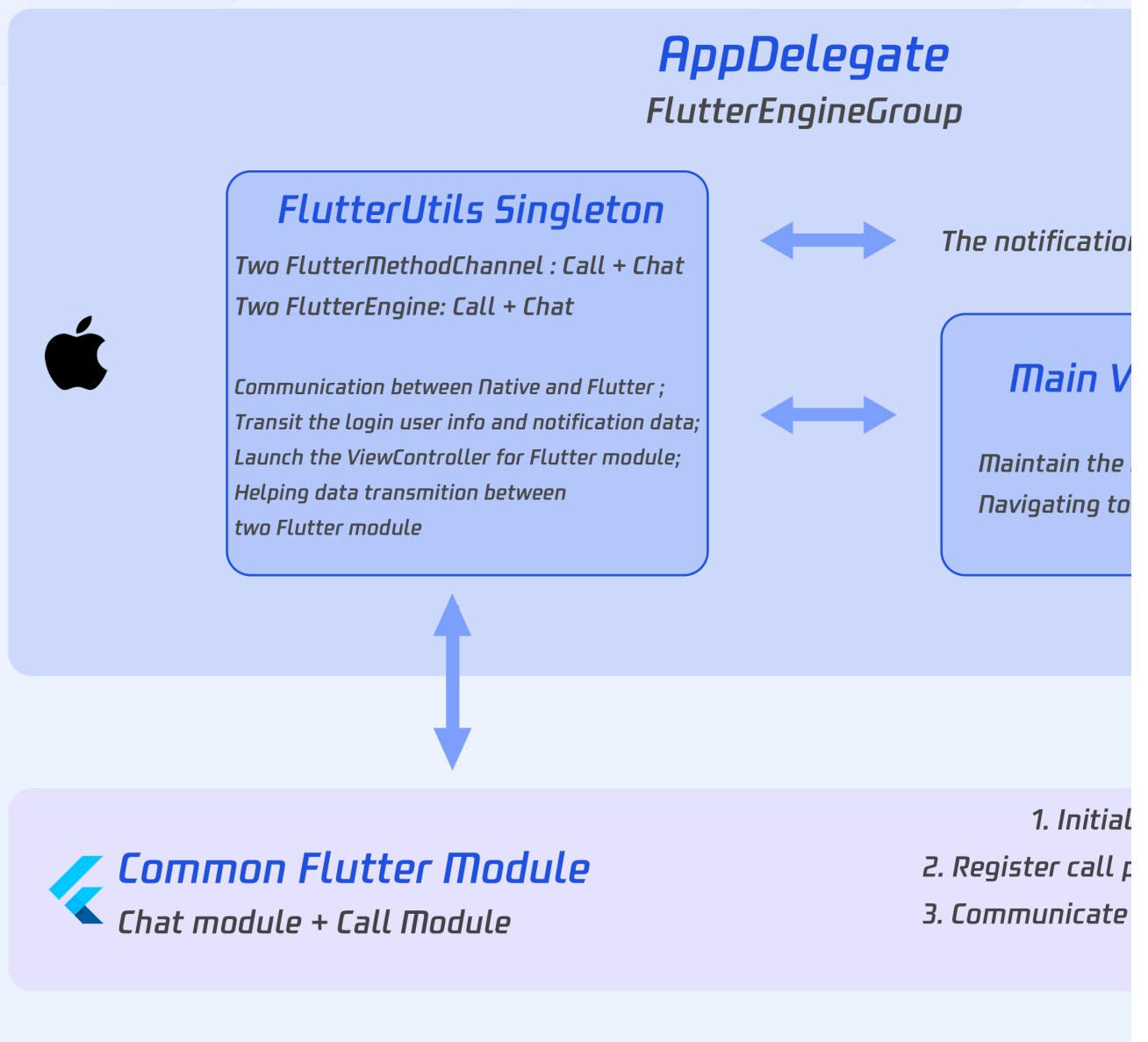
Open your Android project within Android Studio.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing Android app.

FlutterEngineGroup

**Create a `FlutterEngineGroup` to maintain and manage the `FlutterEngine` s.**

The proper place to create a `FlutterEngineGroup` is specific to your host app. As an example, we demonstrate creating a `FlutterEngineGroup` , exposed as a property, on app startup in the app delegate.

Create a new file, `FlutterUtils.kt` , and define a singleton static object `FlutterUtils` .
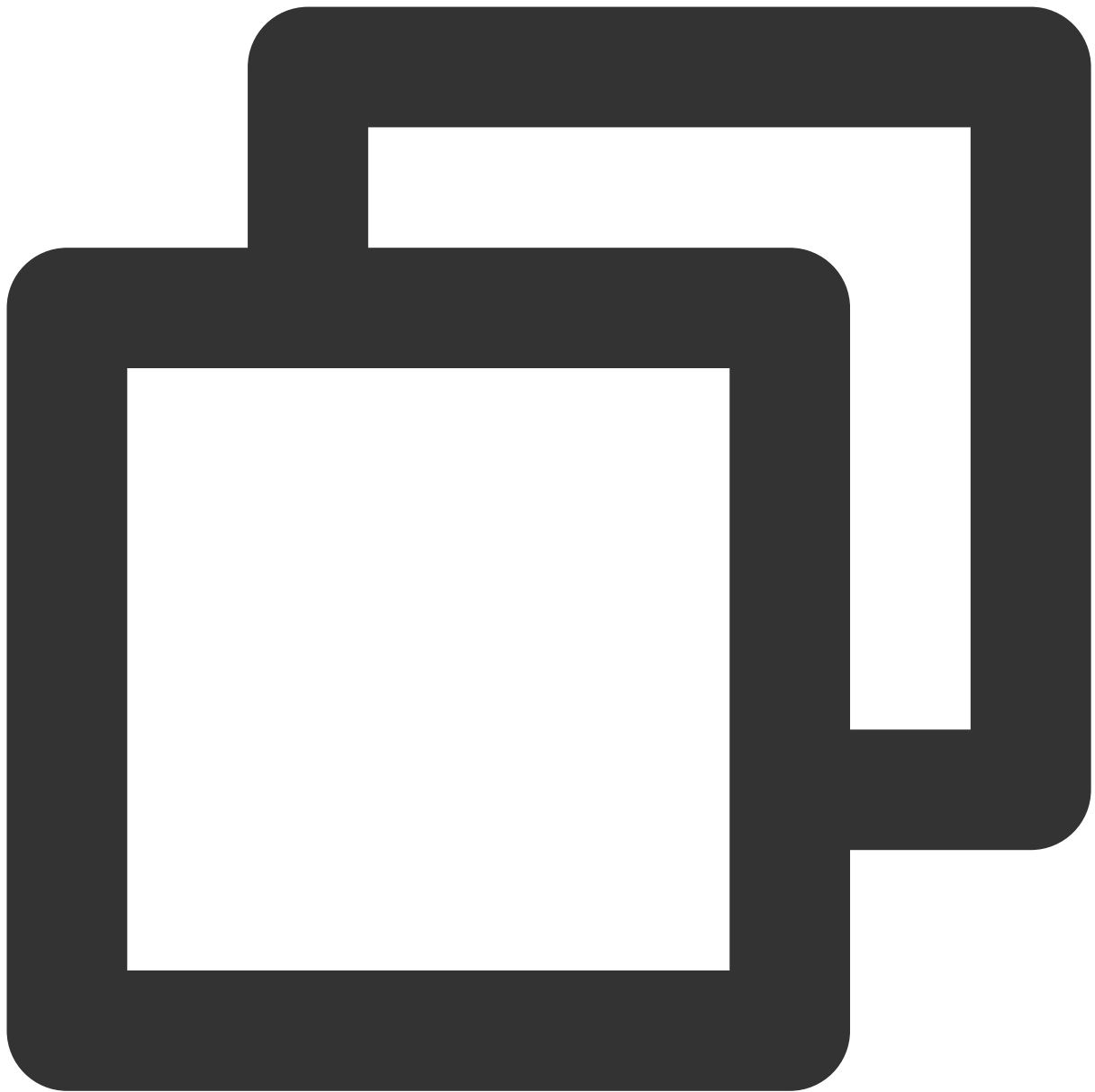
```
@SuppressLint("StaticFieldLeak")
object FlutterUtils {}
```

Create a `FlutterEngineGroup` **to maintain and manage the** `FlutterEngine` **s.**

Define a `FlutterEngineGroup` , `FlutterEngine` s and corresponding `MethodChannel` s in `FlutterUtils.kt` .

```
lateinit var context : Context
lateinit var flutterEngines: FlutterEngineGroup
private lateinit var chatFlutterEngine:FlutterEngine
private lateinit var callFlutterEngine:FlutterEngine

lateinit var chatMethodChannel: MethodChannel
lateinit var callMethodChannel: MethodChannel

// Initialize them
flutterEngines = FlutterEngineGroup(context)
...
```

Further developed for this singleton static object

The basic implementation logic of the sample app is that, using a new navigator for the `Activity` for both Chat and Chat.

The `Activity` for Chat is entered and exited by the user, while the `Activity` for Call has been entered and exited automatically, triggered by the listener or making a call manually.

Mainly focus on:

fun init(): Initialize each Flutter instance, register method channel events.

func reportChatInfo(): Reports the current user login info and SDKAppID to the Flutter module to initialize and log in to the Tencent Cloud Chat SDK.

fun launchCallFunc(): Present the `Activity` for Call module, invoked when new call income or user active it manually from Chat module.

fun triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

You can refer to the sample app source code for this object.

Initialize the singleton static object above from the main entry `MyApplication` .

Transit the global context to the singleton static object, and initialize it from `MyApplication.kt` .

```
class MyApplication : MultiDexApplication() {

    override fun onCreate() {
        super.onCreate()
        FlutterUtils.context = this // new
        FlutterUtils.init()         // new
    }
}
```

Listen for and forward the notification click event

Only transit of the data of notification after clicking is necessary as, the initialization of Push plug-in, uploading token and the navigating for notification clicking events have been done in Flutter Chat module.

The reason why we need to do this is the clicking event has been consumed by Android Kotlin, so it is impossible for the Flutter Push plug-in to receive this event.

Note：

Due to the diversity and inconsistency among different manufacturers, we only take OPPO as an example. For the whole manufacturer's support, please refer to this documentation.

Add a new push certificate to the Tencent Cloud Chat console, Select **Open specified in-app page > activity** for the opening method and enter an activity to receive the notification clicking event with EXT data, it's suggested to set it as the home page or the main entrance. Like, we set `MainActivity` for our sample app, `com.tencent.chat.android.MainActivity` .

Add the following code to the Activity you set in the console in the previous step.

The EXT data of the notification can be found from `Bundle` when the `Activity` has been launched by the device, when the user clicks the notification.

You can receive the EXT from `Activity` , and transit them to Flutter.

You can refer to the source code of the sample app for this capability.
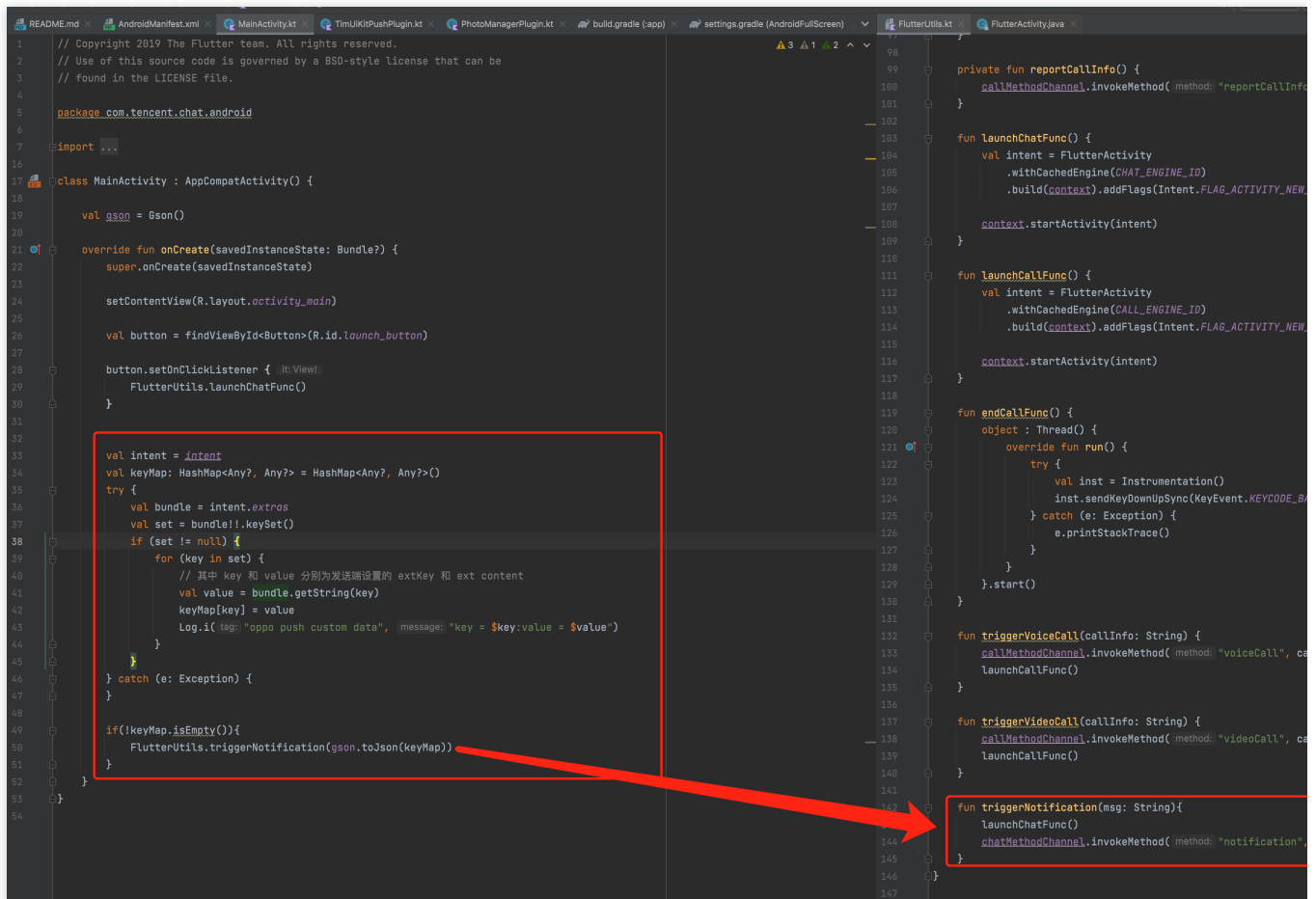
Now, we finished the implementation for Android.

Solution B: Single FlutterEngine

In this solution, the Chat module and Call module embed in one single Flutter instance.
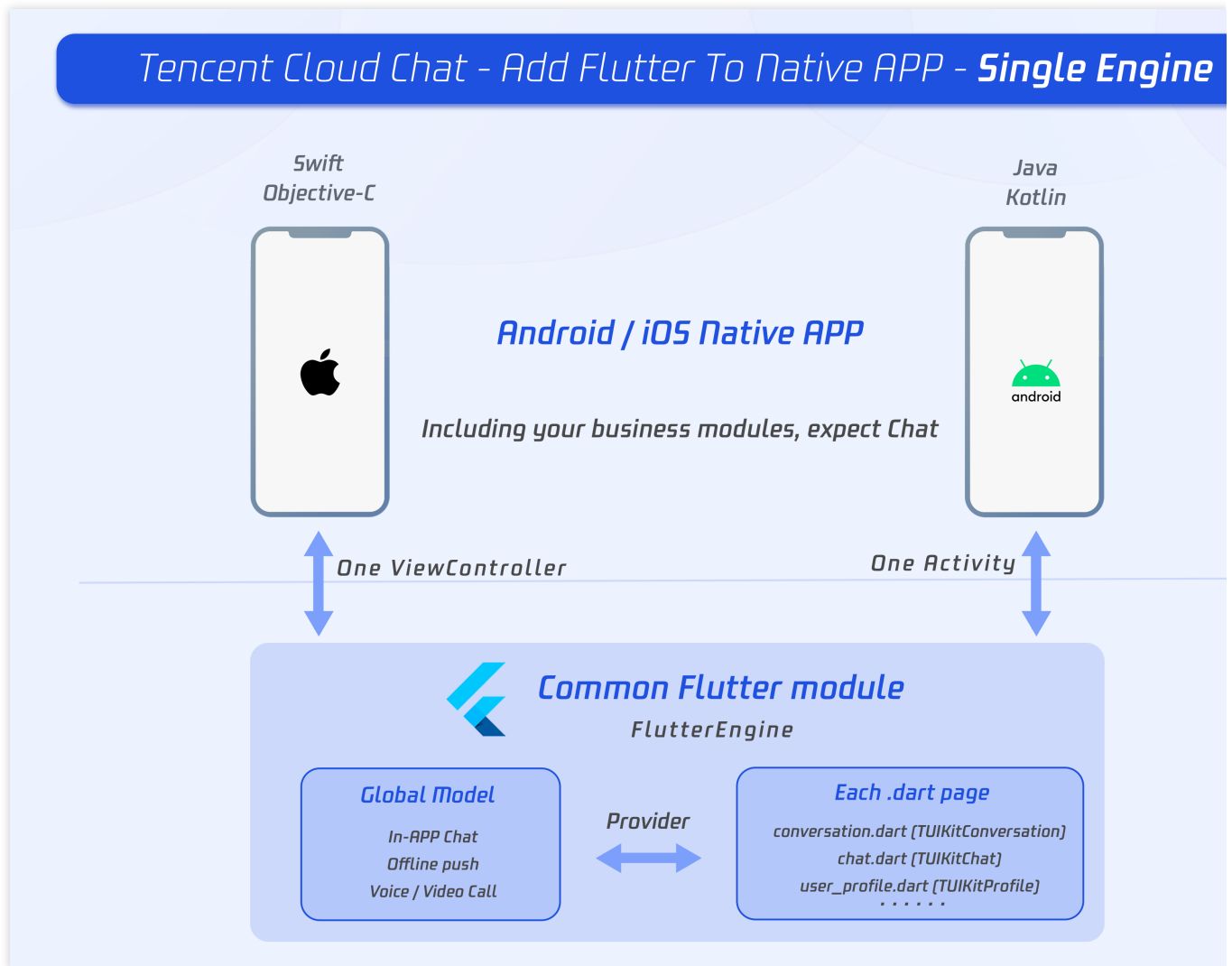
As a result, those modules can only be shown or hidden at the same time.

You can refer to the sample code from GitHub repo this module.

Flutter Module development

To embed Flutter into your existing application, first create a Flutter module.

From the command line, run:
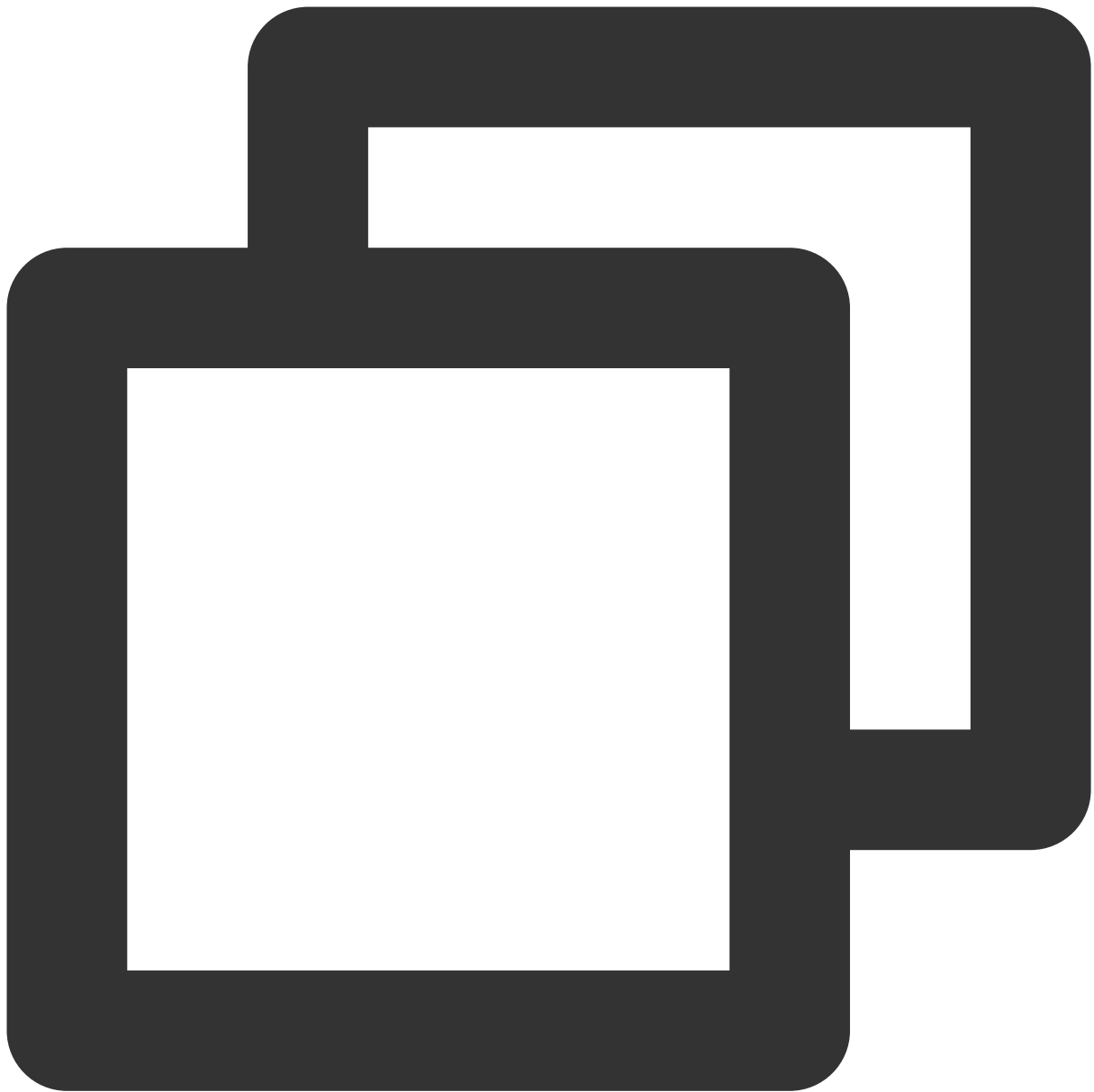
```
cd some/path/
flutter create --template module tencent_chat_module
```

A Flutter module project is created at `some/path/tencent_chat_module/` . From that directory, you can run the same flutter commands you would in any other Flutter project, like `flutter run --debug` or `flutter build ios` . You can also run the module in Android Studio/IntelliJ or VS Code with the Flutter and Dart plugins. This project contains a single-view example version of your module before it's embedded in your existing application, which is useful for incrementally testing the Flutter-only parts of your code.

The `tencent_chat_module` module directory structure is similar to a normal Flutter application:

```
tencent_chat_module/
├── .ios/
│   ├── Runner.xcworkspace
│   └── Flutter/podhelper.rb
├── lib/
│   └── main.dart
├── test/
└── pubspec.yaml
```

Now, we can code within `lib/` .

main.dart

Modify `main.dart` , integrating TUIKit, Offline Push plug-in and Call Plug-in.

The global state, method channel and our Tencent Cloud Chat SDKs, maintained by `ChatInfoModel` .

After receiving the login user info from Native, invoke `_coreInstance.init()` and
`_coreInstance.login()` to initialize and login the SDK. Also, Call plug-in and Push plug-in need to be
initialized.

Note:

Please configure the offline push before uploading the token and use this capability, referring to this documentation.

Tips for Call plug-in:

When a new call invitation is received, call the native method to check if the user is currently on the Flutter page. If not,
force the page to redirect to this module to display the incoming call page.

Tips for Push plug-in:

The callback event of notification clicking is passed from the native layer and used to navigate to the corresponding
chat from EXT data.

Also, this is used as the home page of the chat module. It shows the loading status before logged in, followed by the
conversation list.

In addition, the current status of the application needs to be reported to the Tencent Cloud Chat backend upon each
foreground/background switch from here. For details, refer to this document.

Detailed implementation can refer to the sample code from GitHub repo.

Other widgets from TUIKit

1. Create a file, `push.dart` , used for maintaining the offline push plugin. Detailed implementation can refer to the
sample code from GitHub repo.

2. Create a file, `conversation.dart` , used to implement group profile widget `TIMUIKitGroupProfile` .
Detailed implementation can refer to the sample code from GitHub repo.

3. Create a file, `user_profile.dart` , used to implement the user profile widget `TIMUIKitProfile` .
Detailed implementation can refer to the sample code from GitHub repo.

4. Create a file, `group_profile.dart` , used to implement group profile widget `TIMUIKitGroupProfile` .
Detailed implementation can refer to the sample code from GitHub repo.

5. Create a file, `chat.dart` , used for implementing the history message list and sending messages widget
`TIMUIKitChat` . This page can also navigate to `user_profile.dart` and
`conversation.dart` .Detailed implementation can refer to the sample code from GitHub repo.

At this point, the Flutter module has been developed.

iOS Native development

Here, we take `Swift` as an example, while `Objective-C` is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your iOS project within XCode.

If your existing application (MyApp) doesn't already have a `Podfile` , follow the CocoaPods getting started guide to add a Podfile to your project.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing iOS app.

FlutterEngine

Create a FlutterEngine.

The proper place to create a `FlutterEngine` is specific to your host app. As an example, we demonstrate creating a `FlutterEngine` , exposed as a property, on app startup in the app delegate.

```
import UIKit
import Flutter
import FlutterPluginRegistrant

@UIApplicationMain
class AppDelegate: FlutterAppDelegate { // More on the FlutterAppDelegate.
 lazy var flutterEngine = FlutterEngine(name: "tencent cloud chat")

  override func application(_ application: UIApplication, didFinishLaunchingWithOpti
    // Runs the default Dart entry point with a default Flutter route.
    flutterEngine.run();
```

```
        GeneratedPluginRegistrant.register(with: self.flutterEngine);
        return super.application(application, didFinishLaunchingWithOptions: launchOptio
    }
}
```

Create a singleton static object to manage the FlutterEngine.

This singleton is used for managing `FlutterEngine` in one place, and provides methods to the whole project to invoke methods related to the Flutter module.

The basic implementation logic of the sample app is that, using a new navigator for the ViewController of Flutter module, and show or hidden can be handled automatically according to call.

Create a new file, `FlutterUtils.swift` , and coding, refer to our sample app source code.

Mainly focus on:

private override init(): Initializes each Flutter instance, registers method channel, and listens for events.

func reportChatInfo(): Report the current user info to the Flutter module, for initialization and login Tencent Cloud Chat SDK.

func launchChatFunc(): Present the ViewController for Flutter module.

func triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

Listen for and forward the notification click event

Only transit of the data of notification after clicking is necessary as, the initialization of Push plug-in, uploading token and the navigating for notification clicking events have been done in Flutter Chat module.

The reason why we need to do this is the clicking event has been consumed by iOS Swift, so it is impossible for the Flutter Push plug-in to receive this event.

Add the following codes to `AppDelegate.swift` .

```
19    }
20
21    @UIApplicationMain
22    class AppDelegate: FlutterAppDelegate {
23      lazy var flutterEngines = FlutterEngineGroup(name: "chat.flutter.tencent", project: nil)
24
25      override func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
26
27          if #available(iOS 10.0, *) {
28            UNUserNotificationCenter.current().delegate = self
29          }
30
31          if let remoteNotification = launchOptions?[UIApplication.LaunchOptionsKey.remoteNotification]{
32              let notificationExt: [AnyHashable:Any] = remoteNotification as! [AnyHashable:Any]
33              let remoteNotificationString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
34              FlutterUtils.shared.triggerNotification(msg: remoteNotificationString)
35          }
36
37          return super.application(application, didFinishLaunchingWithOptions: launchOptions);
38      }
39
40      override func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse, withCompletionHandler completio
            @escaping () -> Void) {
41              let notificationExt: Dictionary = response.notification.request.content.userInfo
42              let notificationExtString: String = notificationExt.jsonStringRepresentaiton ?? "{}"
43              FlutterUtils.shared.triggerNotification(msg: notificationExtString)
44          }
45
46      }
```

At this point, the implementation for iOS is complete.

Android Native Development

Here, we take `Kotlin` as an example, while `Java` is also available.

Note:

The following structure and code is for demonstration purposes only, you could modify it to meet your actual needs dynamically.

Open your Android project within Android Studio.

Import Flutter Module

Please refer to this part, adding the Flutter module to your existing Android app.

FlutterEngine

Create a singleton static object to manage the FlutterEngine.

This singleton is used for managing `FlutterEngine` in one place, and provides methods to the whole project to invoke methods related to the Flutter module.

Create a new file, `FlutterUtils.kt`, and define a singleton static object `FlutterUtils`.

```
@SuppressLint("StaticFieldLeak")
object FlutterUtils {}
```

Create a `FlutterEngine`.

Define a `FlutterEngine` and corresponding `MethodChannel` in `FlutterUtils.kt`.

```
lateinit var context : Context
private lateinit var flutterEngine:FlutterEngine



flutterEngine = FlutterEngine(context)
```

Further developed for this singleton static object

The basic implementation logic of the sample app is that, using a new navigator for the Activity for both Chat and

Chat.

Mainly focus on:

fun init(): Initialize each Flutter instance, register method channel events.

func reportChatInfo(): Reports the current user login info and SDKAppID to the Flutter module to initialize and log in to the Tencent Cloud Chat SDK.

fun launchChatFunc(): Present the `Activity` for Flutter module.

fun triggerNotification(msg: String): Transit the data of notification, after the user clicks it, and Chat module may navigate to the corresponding chat page.

Detailed implementation can refer to the sample code from GitHub repo.

Initialize the singleton static object above from the main entry `MyApplication`.

Transit the global context to the singleton static object, and initialize it from `MyApplication.kt`.

```
class MyApplication : MultiDexApplication() {

    override fun onCreate() {
        super.onCreate()
        FlutterUtils.context = this // new
        FlutterUtils.init()         // new
    }
}
```

Listen for and forward the notification click event

Only transit of the data of notification after clicking is necessary as, the initialization of Push plug-in, uploading token and the navigating for notification clicking events have been done in Flutter Chat module.

The reason why we need to do this is the clicking event has been consumed by Android Kotlin, so it is impossible for the Flutter Push plug-in to receive this event.

> Due to the diversity and inconsistency among different manufacturers, we only take OPPO as an example. For the whole manufacturer's support, please refer to this documentation.

Add a new push certificate to the Tencent Cloud Chat console, Select **Open specified in-app page > activity** for the opening method and enter an activity to receive the notification clicking event with EXT data, it's suggested to set it as the home page or the main entrance. Like, we set `MainActivity` for our demo, `com.tencent.chat.android.MainActivity` .

Tencent Cloud

Add the following code to the Activity you set in the console in the previous step.

The EXT data of the notification can be found from `Bundle` when the `Activity` has been launched by the device, when the user clicks the notification.

You can receive the EXT from `Activity` , and transit them to Flutter.

You can refer to the demo source code for this capability.

At this point, the implementation for Android is complete.

Additional solution: Initialize Tencent Cloud Chat from Native SDK

Sometimes, you may prefer to integrate a chat module to your existing UI without a complex chat page.

For example, during a game, you want to let players chat with each other during the match without navigating to the full screen chat page.

Means, you may not wish to launch a complex Flutter engine, before the user switches to the chat page, but hope they can still chat in a small module directly.

In this case, you can initialize and login using the native SDK and build in-app chat features.

Note:

However, you can also choose to initialize and login within Flutter up to your needs. This process should only be executed once, no matter where you execute it.

It's unnecessary to import Native SDK manually, as our Flutter SDK can help you integrate it.

Initialize and login

iOS Swift is used as an example to demonstrate how to initialize and log in with the native SDK.

```
import ImSDK_Plus


func initTencentChat(){
    if(isLoginSuccess == true){
        return
    }
    let data = V2TIMManager.sharedInstance().initSDK( Yours SDKAPPID , config: n
    if (data == true){
        V2TIMManager.sharedInstance().login(
            chatInfo.userID,
```

```
            userSig: chatInfo.userSig,
            succ: {
                self.isLoginSuccess = true
                self.reportChatInfo()
            },
            fail: onLoginFailed()
        )
    }
}
```

After that, you could use the API provided by Native SDK to implement your chat modules to your existing UI page manually.

For more information about the Native SDK, please refer to [this documentation](#).

Initialize Flutter TUIKit

After initialization and login from the native SDK, the user info should be provided to Flutter TUIKit by invoking _coreInstance.setDataFromNative().

```
final CoreServicesImpl _coreInstance = TIMUIKitCore.getInstance();
_coreInstance.setDataFromNative(userId: chatInfo?.userID ?? "");
```

You can refer to the sample code from [GitHub repo](#) this module.

This completes the tutorial for using hybrid development with Flutter to integrate Tencent Cloud Chat to your existing app.

If you have any questions, feel free to contact us.

[Telegram Group](#)

[WhatsApp Group](#)

Reference