DUIX.AI Open Source Application Development Handbook

Introduction:

This document aims to provide developers with a complete example of the integration process based on the DUIX.AI open source project. Throughout the development process, you can refer to the configuration, optimization, and considerations of each link. It is hoped that this document will bring convenience to your development work and jointly promote the application and development of the DUIX.AI project.

I. Development Environment

Item	Description
System	MacOS 13.6.6
CPU Architecture	M2 ARM64
Development IDE	Android Studio Preview 2024.1

II. Android Environment

1. Build Tools

Item	Version
Gradle	8.7

2. SDK Platforms

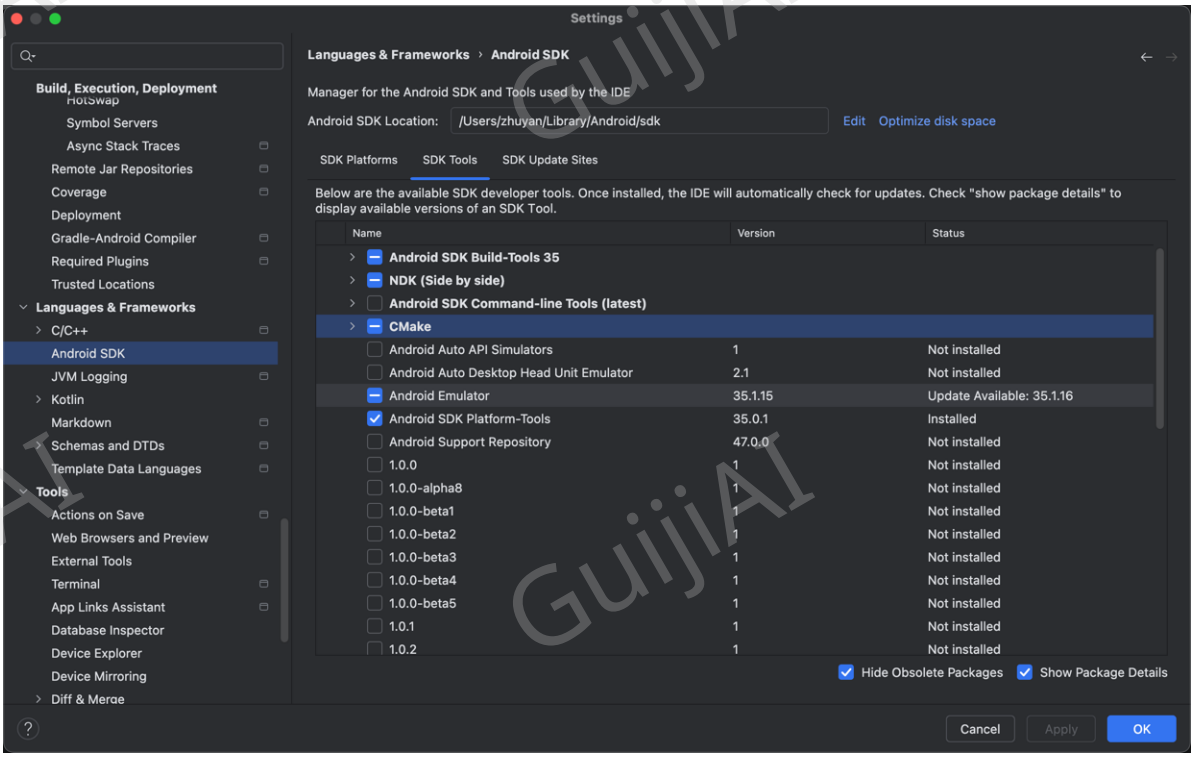
Item	Version
Android API 35	35.0.0
Android 9.0 (“Pie”)	9.0

3. SDK Tools

Item	Version
Android SDK Build-Tools 35	35.0.0

Item	Version
NDK (side by side)	27.0.11902837 rc2
CMake	3.22.1
Android Emulator	35.1.15
Android SDK Platform-Tools	35.0.1

Environment installation as follows:



Attached is the Android environment installation and configuration tutorial

https://blog.csdn.net/a910247/article/details/138012201

III. Project Preparation

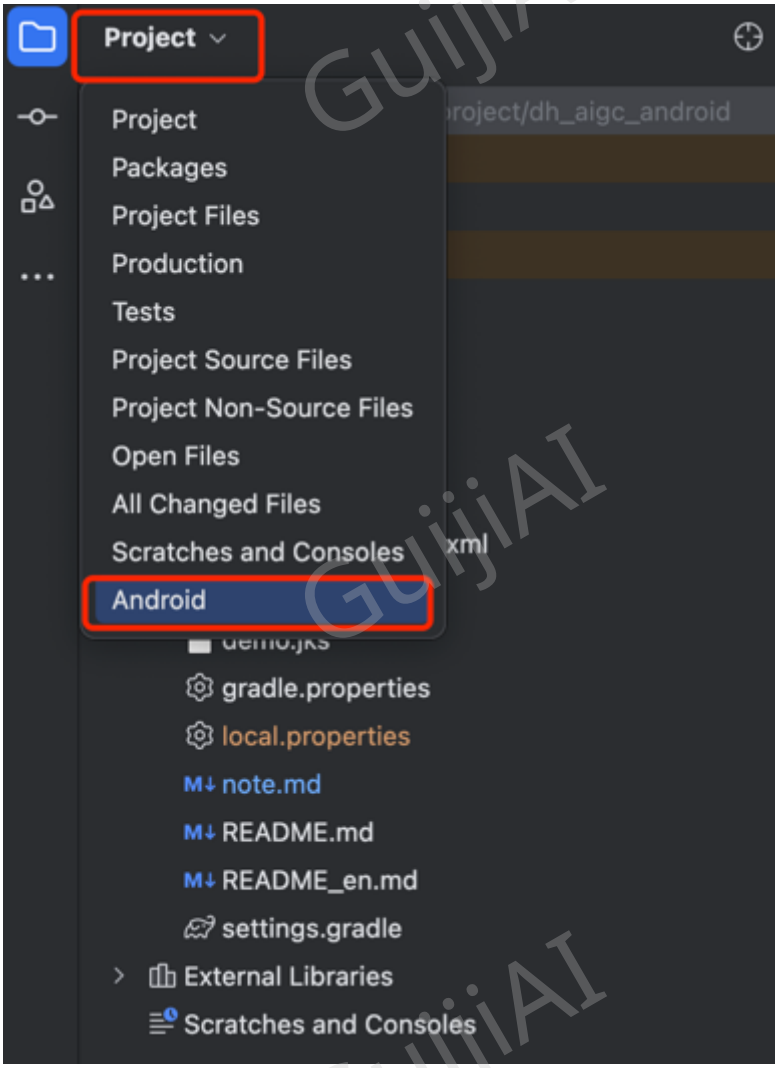
Download address：https://github.com/GuijiAI/duix.ai

Git Clone address：https://github.com/GuijiAI/duix.ai.git

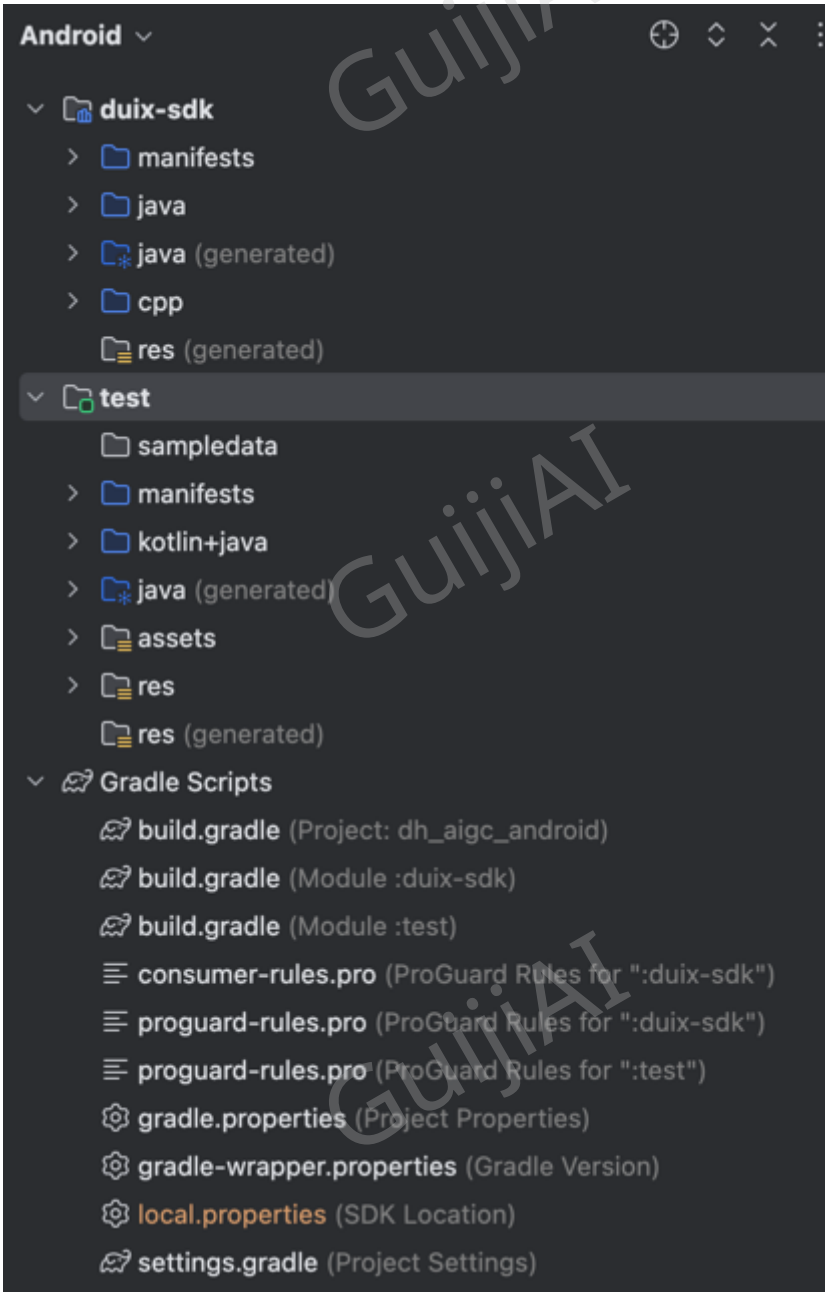
Import the project dh_aigc_android using Android Studio

Wait for the Gradle environment and dependent packages to download...

After the dependencies are downloaded, switch to the Android view



After switching, the project structure is as follows



The project contains two sub-projects

1. duix-sdk Project

duix digital human rendering source code, as a dependency package for the test project

build.gradle (Module: duix-sdk) Configuration as follows

```
plugins {
    id 'com.android.library'
}
```

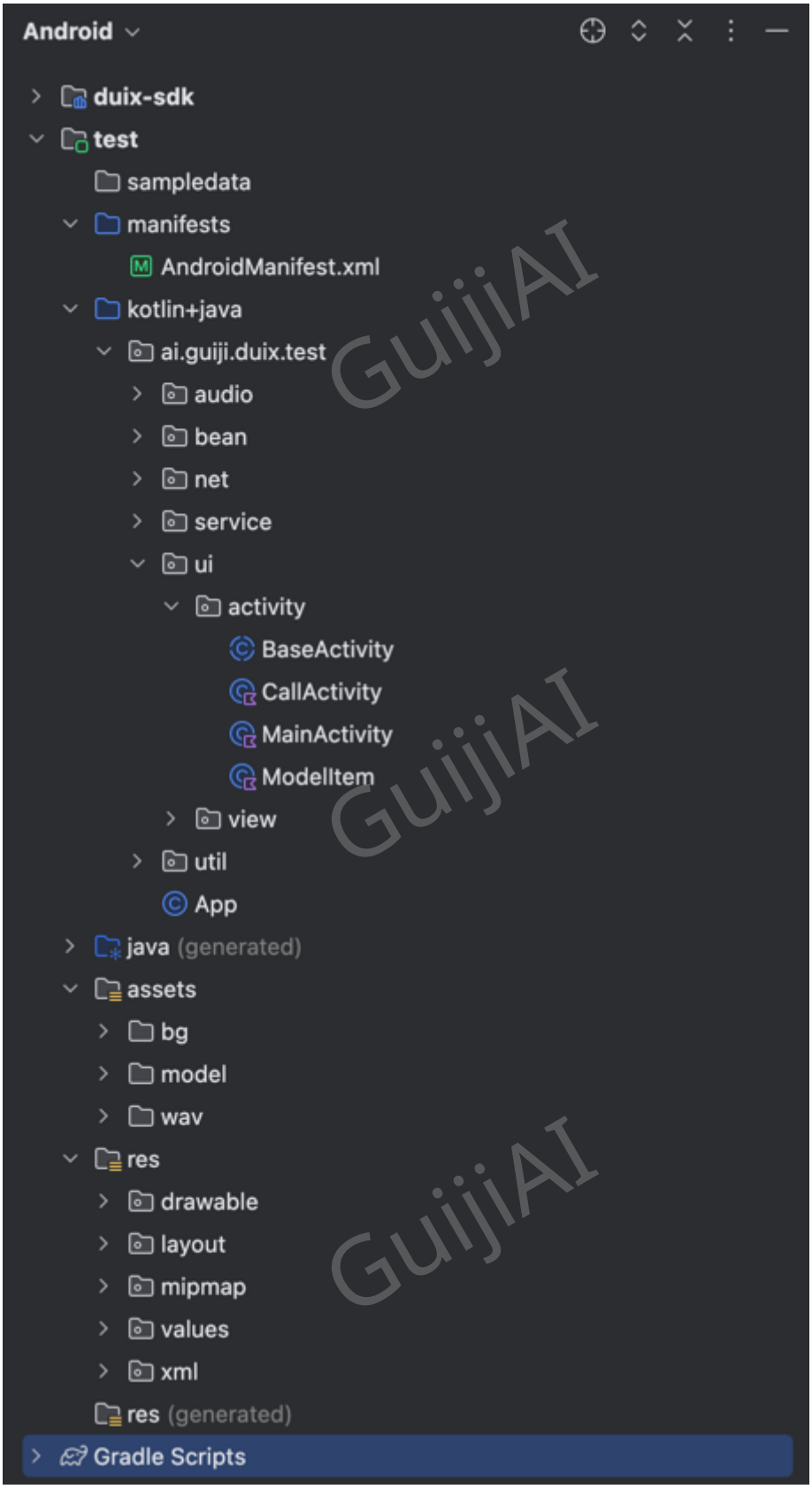
We don't need to touch the duix-sdk source code here, as the demo project dependency has already been referenced for us.

You can view the reference in test.build.gradle

```
implementation project(":duix-sdk")
```

2. test Project

android application demo project, this is also the app project we need to develop, the project structure is as follows:



build.gradle (Module: test) Configuration as follows

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}
```

Add dependencies in build.gradle (Module: test)

```
dependencies {
    // Reference SDK project dependency
    implementation project(":duix-sdk")
    // The SDK uses exoplayer to handle audio (required)
    implementation 'com.google.android.exoplayer:exoplayer:2.14.2'

    // HTTP request dependency
    implementation 'com.squareup.okhttp3:okhttp:4.10.0'
    implementation 'com.squareup.okhttp3:okhttp-sse:4.10.0'
    ...
}
```

Android permission requirements, add the following configuration in AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Recording permission -->
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <!-- Storage permission -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

</manifest>
```

3、 Activity

Here we only introduce the most basic Activity, other Activities please implement by yourself.

Each Activity corresponds to an xml ui configuration, you can define ui interface controls

BaseActivity

```
Defines a basic abstract Activity as the base class for rendering, mainly
implementing onCreate(), onDestroy(), onPause(), onResume() and other lifecycle
methods.
```

MainActivity

```
The main Activity of the app, mainly implementing the download and loading of the digital human image resources and the completion check of the loaded.
```

CallActivity

```
Loading and rendering of digital human images, mainly implementing loading, rendering, playing, pausing, destroying and other operations of digital human images.
```

IV. SDK Call and API Description

1. Digital Human Image Construction

Use the RenderSink interface to accept rendering frame data, the SDK provides an implementation of DUIXRenderer.java for this interface. You can also implement the interface yourself to customize rendering.

The definition of RenderSink is as follows:

Use DUIXRenderer and DUIXTextureView control to simply implement rendering display, this control supports transparent channels and can freely set the background and foreground:

```
    // ...
    mDUIXRender =
        DUIXRenderer(
            mContext,
            binding.glTextureView
        )

    binding.glTextureView.setEGLContextClientVersion(GL_CONTEXT_VERSION)
    binding.glTextureView.setEGLConfigChooser(8, 8, 8, 8, 16, 0) // Transparent
    binding.glTextureView.isOpaque = false           // Transparent
    binding.glTextureView.setRenderer(mDUIXRender)
    binding.glTextureView.renderMode = GLSurfaceView.RENDERMODE_WHEN_DIRTY
// Must be called after setting Render
```

2. duix Object Construction

Build the DUIX object and add callback events in the rendering page onCreate() phase

```
duix = DUIX(mContext, baseDir, modelDir, mDUIXRender) { event, msg, info ->
    when (event) {
        ai.guiji.duix.sdk.client.Constant.CALLBACK_EVENT_INIT_READY -> {
            initOK()
        }

        ai.guiji.duix.sdk.client.Constant.CALLBACK_EVENT_INIT_ERROR -> {
```

```
        }
        // ...

    }
}
// Asynchronous callback result
duix?.init()
```

DUIX object construction description:

Parameter	Type	Description
context	Context	System context
baseDir	String	The configuration file where the model drive is stored needs to be managed by yourself. You can unzip the compressed file to external storage and provide the folder path
modelDir	String	The folder where the model file is stored needs to be managed by yourself. You can unzip the compressed file to external storage and provide the folder path
render	RenderSink	Rendering data interface, the SDK provides a default rendering component that inherits from this interface, or you can implement your own
callback	Callback	Various callback events handled by the SDK

Refer to the ai.guiji.duix.test.ui.activity.CallActivity example in test

3. Start Digital Human Broadcast

After initialization is successful, you can play audio to drive the image

```
duix?.playAudio(wavPath)
```

Parameter description:

Parameter	Type	Description
wavPath	String	Local WAV file with 16k sample rate, single channel, 16-bit depth

4. Terminate Current Broadcast

When the digital human is broadcasting, call this interface to terminate the broadcast.

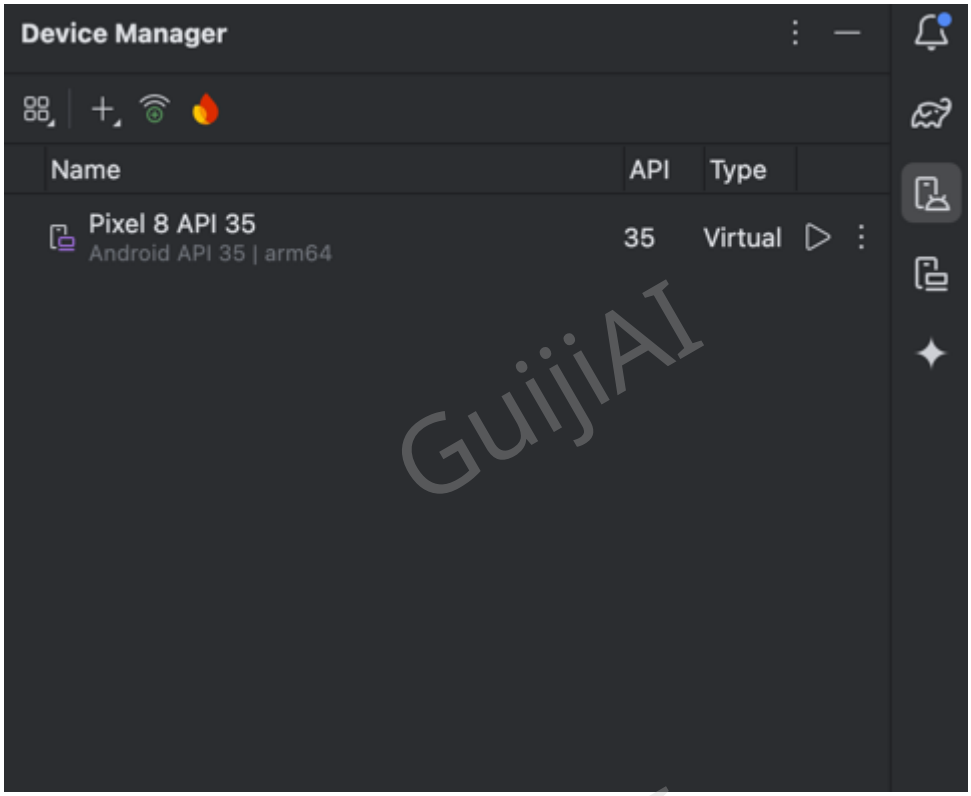
Call example as follows:

```
duix?.stopAudio()
```

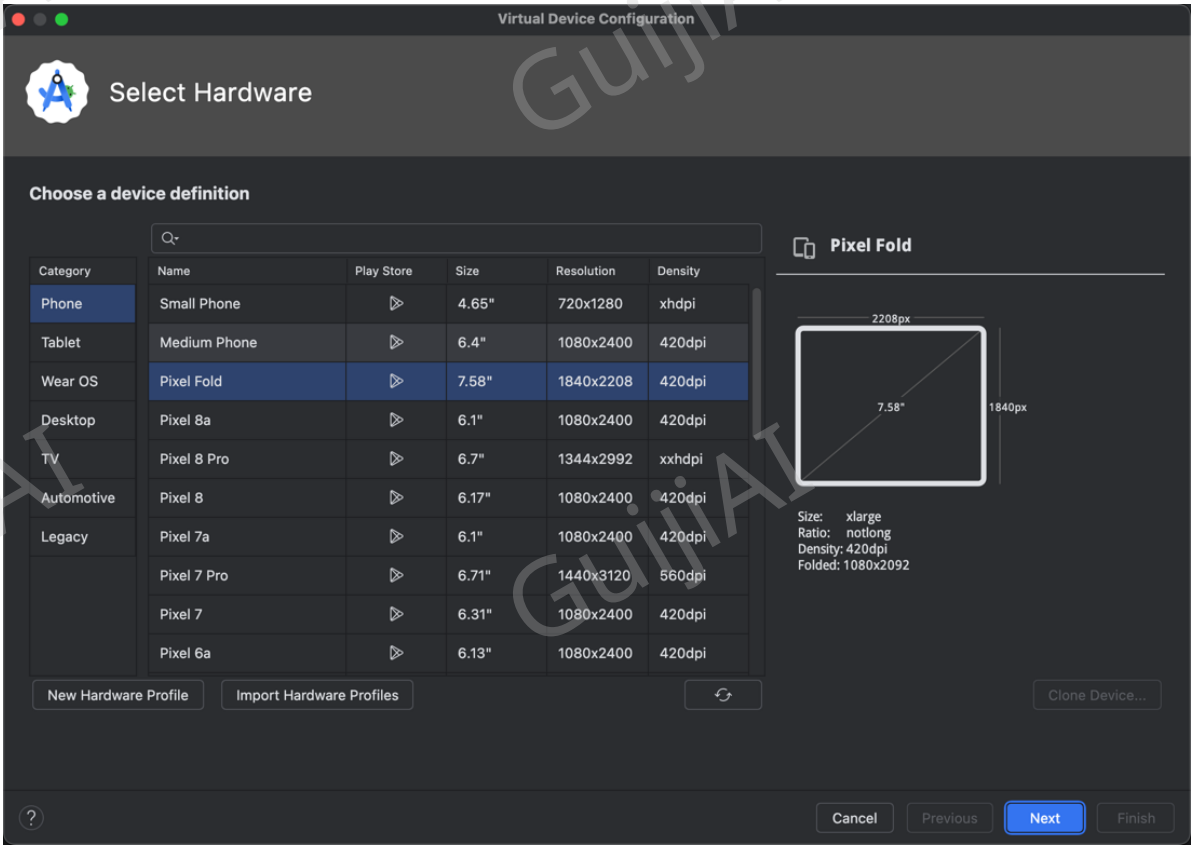
V. Project Operation

1. Create a Virtual Machine

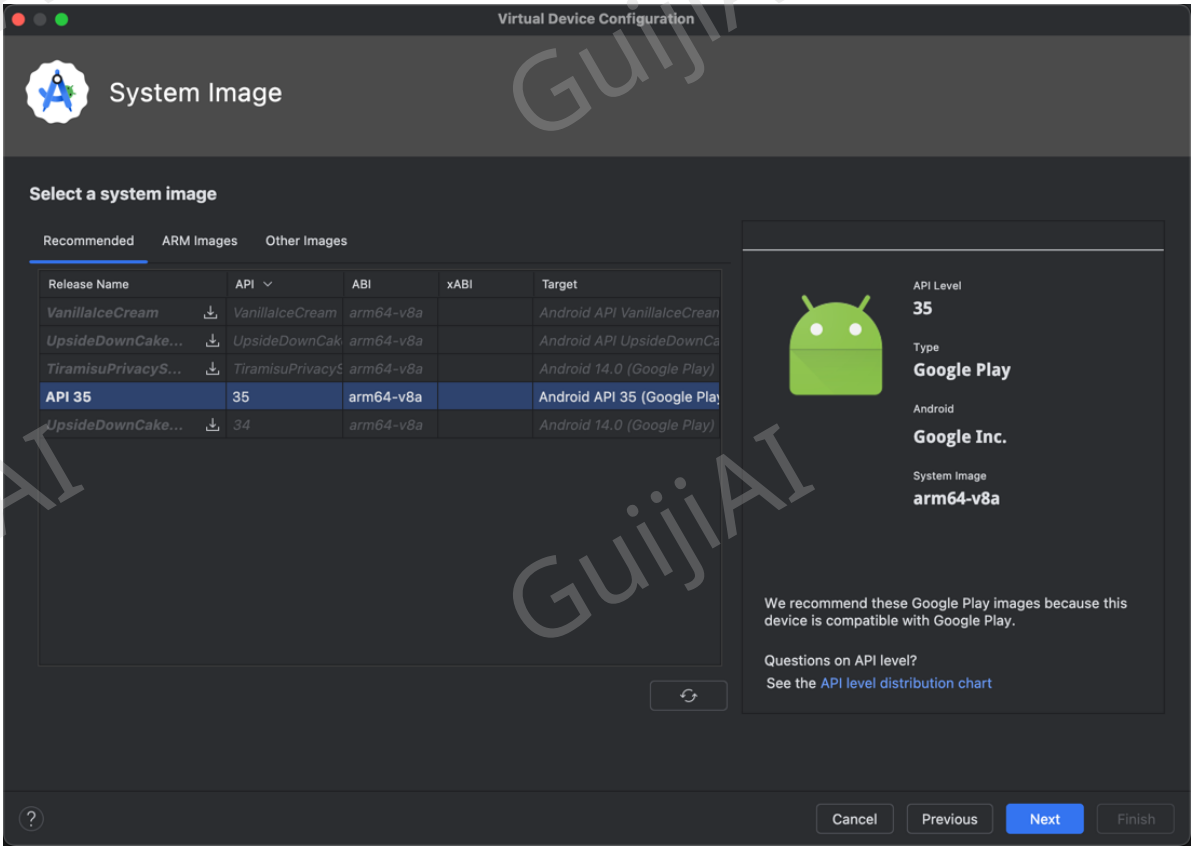
Click + to add a virtual machine



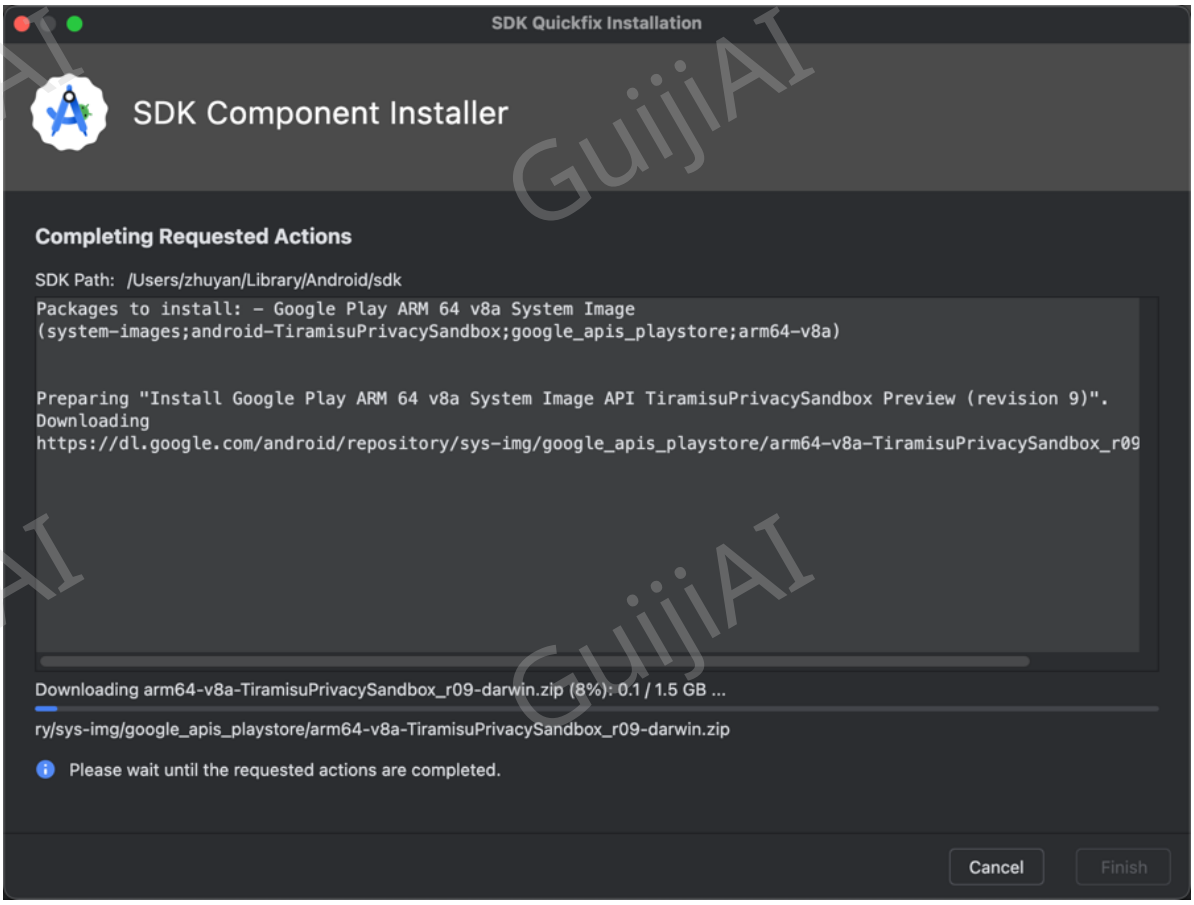
Select a device



Select the virtual machine image, you need to download the image for the first time (select the image that matches your CPU)



Wait for the image to download

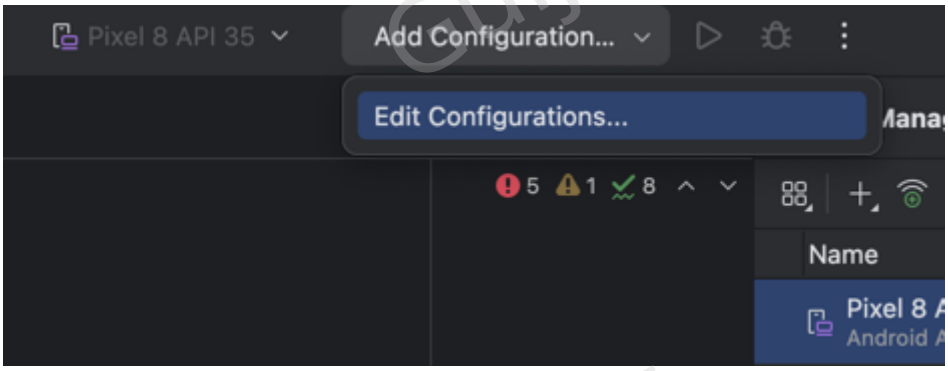


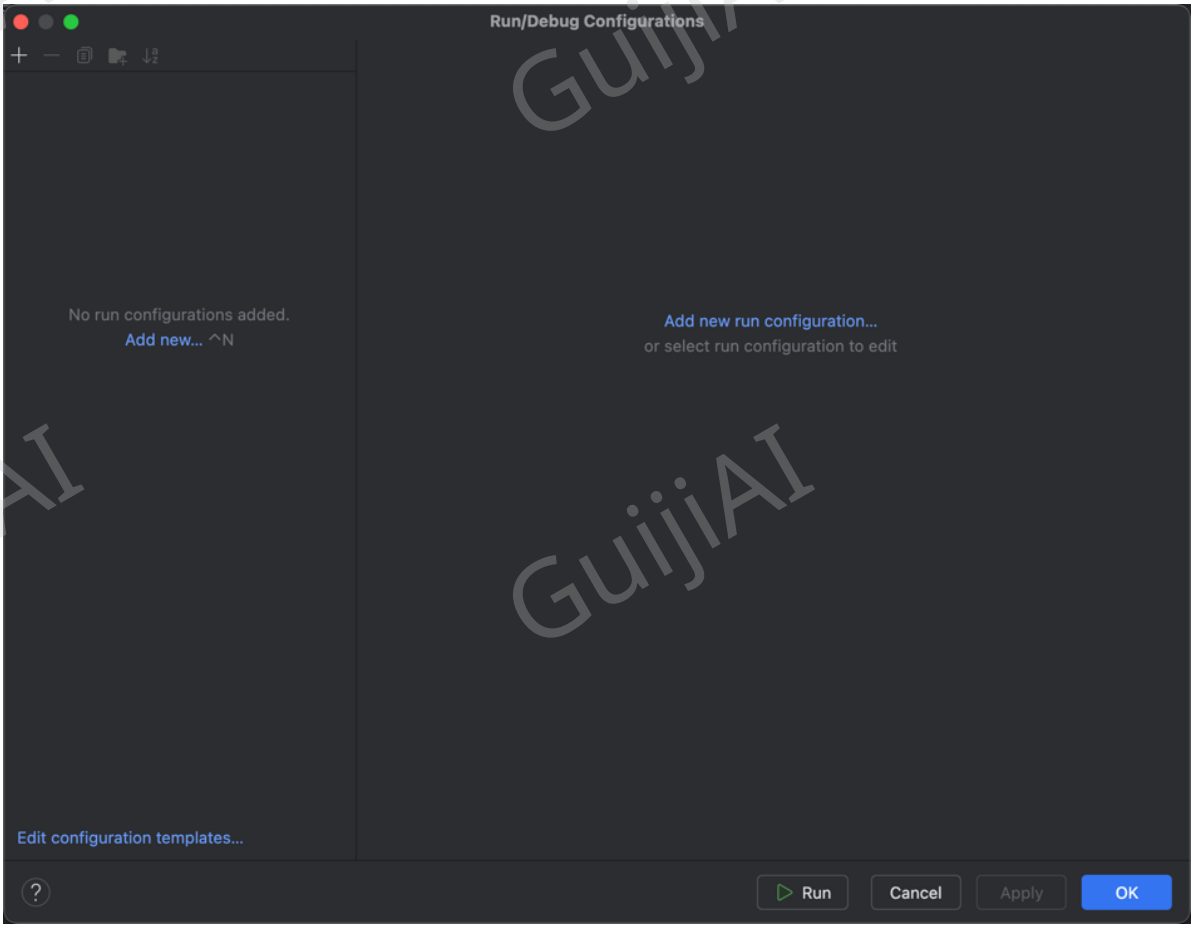
Set virtual machine parameters

Complete virtual machine configuration

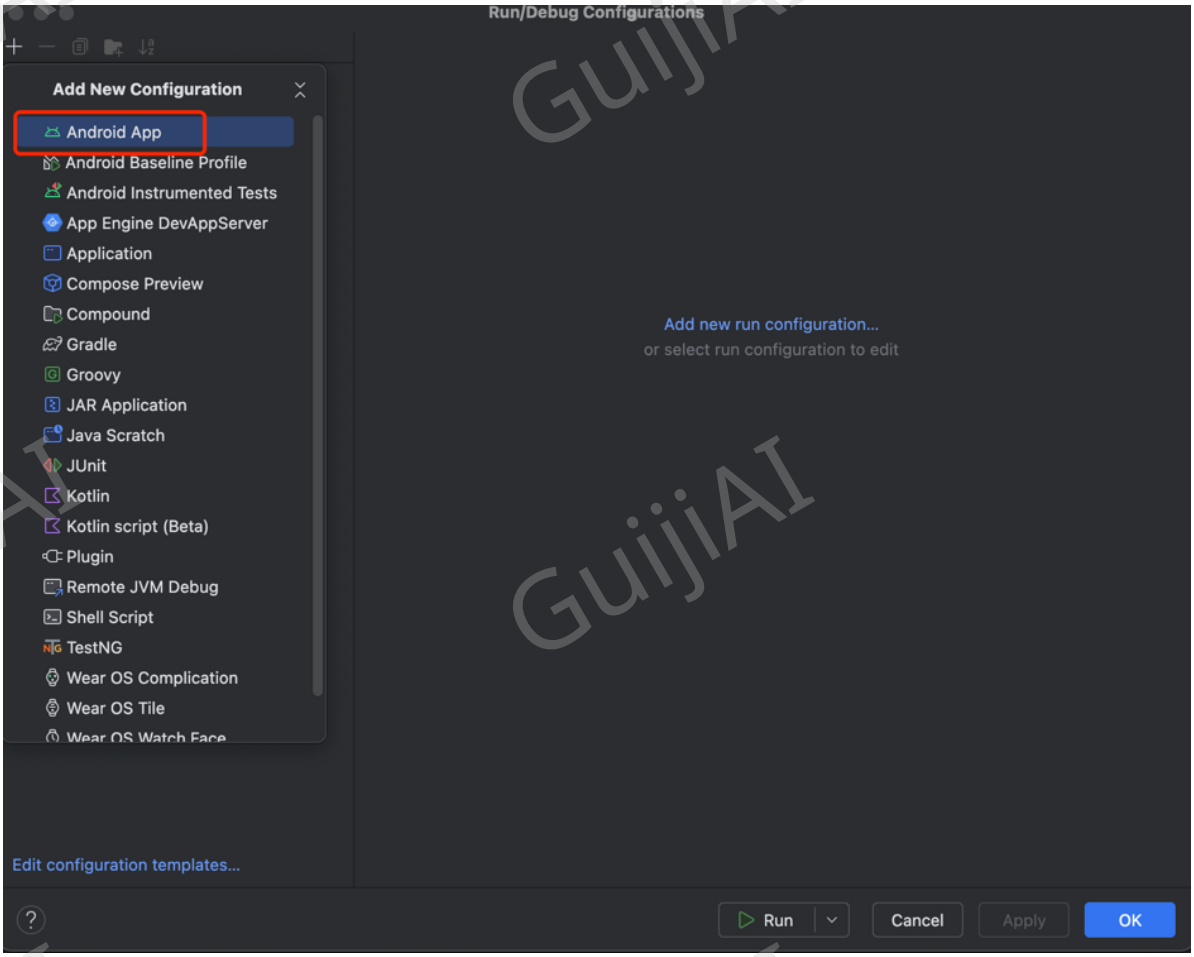
2. Start the Project

Add a Configuration

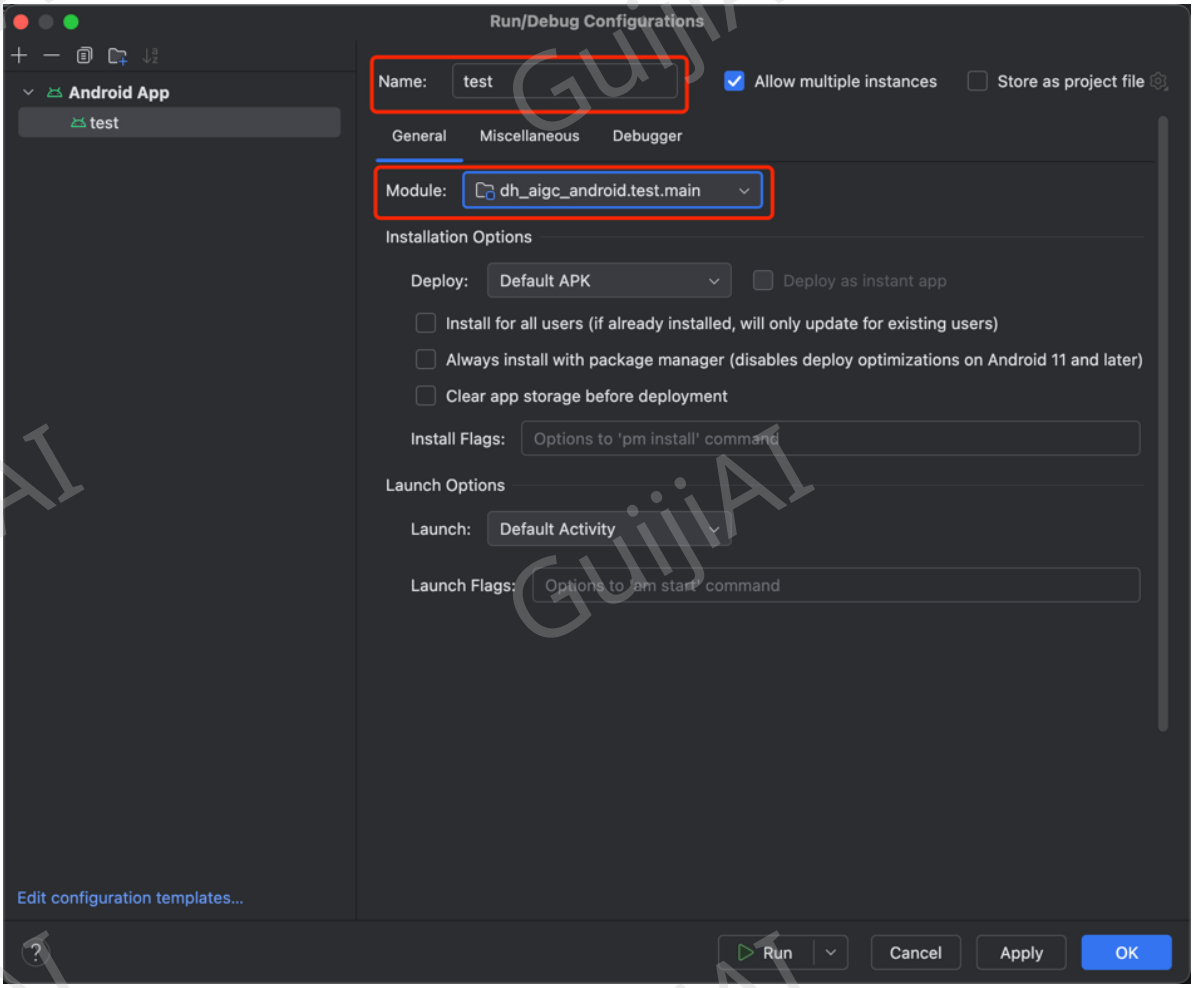




Add an Android APP



Fill in the project name, select Module



Start the project