**Wide KYC SDK**

Wide Technologies provides an SDK for you to implement integration with your native mobile application. This document provides you with an overview of the SDK integration in terms of its architecture, interaction flow, supported products, and general integration process.

# **1. Integration architecture**

Here is the architecture of our SDK integration.

Diagram

Description automatically generated

*Figure 1. SDK integration architecture*

SDK integration consists of two parts:

1. Client-side integration: integrate the Wide SDK into the merchant application. The Wide SDK provides a ready set of screens and tools for both iOS and Android applications to capture required user data such as face images, identity document images, and so on. By integrating the Wide SDK, you can easily create a friendly interaction experience for your users in terms of:
   1. Well designed UI to guide your users through the simple and easy business process
   2. High success rate and high security with multiple algorithms applied
   3. Simplified integration process by directly uploading images to the Wide service to process
2. Server-side integration: Expose endpoints for your (merchant) application in your (merchant) server so that the merchant application can interact with the merchant server, which then accesses the Wide API to initialize a transaction and double-check verification results.

# **2. Supported products**

Wide SDK integration can be applied to the following products:

* Face Capture (Passive Liveness)
* ID Recognize (OCR)
* ID Verification by Population and Civil Registration Agency (Dukcapil)
* RealID (ID Recognize -> Face Capture -> ID Verification)

# **3. Interaction flow**

The following diagram illustrates the whole interaction flow when a WIDE service is started through the mobile application.

Diagram

Description automatically generated

*Figure 2. Sequence diagram*

1. A user initiates a business process (for example, an ID Recognize process) through the merchant application.
2. The merchant app calls the getMetaInfo interface to obtain the meta-information about the Wide SDK and the user's device for preparation.
3. The Wide SDK returns the meta-information to the merchant application.
4. The merchant application initializes a transaction and passes the meta-information and product type to the merchant server.
5. With the meta-information and product info as an input, the merchant server calls the initialize API to obtain the configuration information, which includes parameters about SDK connection and behavior.
6. The Wide server performs a usability check based on the meta-information. If the check is passed, the Wide server returns the configuration information to the merchant server.
7. The merchant server returns the configuration information to the merchant application.
8. The merchant application starts the ZOLOZ SDK with the configuration information that is obtained in Step 7.
9. The Wide SDK interacts with the user, captures required data (for example, ID images), and uploads it to the Wide server for verification. There might be multiple rounds of interaction between the Wide SDK and Wide server.
10. The Wide server performs related checks on the uploaded user data, and returns the transaction status to the Wide SDK. If all the corresponding checks are passed, a result code that indicates success is returned to the Wide SDK; otherwise, the process might be interrupted and further interactions are needed between the user and the Wide SDK.
11. The Wide SDK notifies the merchant application that the transaction is completed.
12. The merchant application syncs with the merchant server that the transaction is completed and starts a double check on the transaction details.
13. The merchant server calls the checkResult API to check the transaction details with the Wide server again.
14. The Wide server returns the transaction details to the merchant server.

Note: To ensure information security, sensitive information such as captured face images is only returned to the merchant server.

1. The merchant server filters the transaction details that are returned from the Wide server and returns the information that is not sensitive to the merchant application.
2. The merchant application informs the user that the process is completed.

# **4. Integration Process**

This section introduces how to implement the Wide SDK on and client-side mobile.

## **4.1 SDK requirements**

The Wide SDK supports both Android and iOS. To integrate the Wide SDK, ensure your mobile device system meets the following requirements:

* Operation system version must be Android 4.3 or later, or iOS 8 or later.
* Permissions of network and camera must be granted to the Wide SDK.

Note: please note x86 architecture is not supported.

## **4.1 Android Integration**

### **4.1.1 Import the SDK**

**a. Configure the maven repository**

Add the following maven repository configuration to the build.gradle file that is in your project root directory.

maven { url 'https://jitpack.io' }

**b. Add the SDK dependency**

Add the SDK as a dependency in your module's (application-level) gradle file (usually app/build.gradle).

implementation 'com.google.code.gson:gson:2.8.9'

implementation ('com.github.wide-mobile:widekyc:1.0.1@aar'){transitive = true}

### **4.1.2. Get meta information**

Use the `WKYC` class and its method `getMetaInfo` for transaction later.

WKYC wkyc = WKYC.getInstance(applicationContext);

String metaInfo = WKYC.getMetaInfo(applicationContext);

### **4.1.3. Initialize a transaction**

Send a request that contains the meta-information to your (merchant) server to initialize a transaction. Then your (merchant) server needs to call the initialize API to obtain the configuration and return it to your (merchant) application.

### **4.1.4 Start the transaction flow**

Construct the `WKYCRequest` object with the config that is returned from the merchant server using the GSON library to easily map the result to `WKYCConfig` class

InitResponse initResponse = new Gson().fromJson(configurationJson.getString(Constants.CONTENT), InitResponse.class);

WKYCRequest request = new WKYCRequest();

request.wkycConfig = new Gson().fromJson(resultJson.getString("content"), WKYCConfig.class);

request.moduleConfig.put(ZLZConstants.LOCALE, locale);

return request;

Start the transaction flow by calling the start method with the `WKYCRequest` object that is constructed in Step 4(a). You also need to override the callback functions to handle the transaction result.

wkyc.start(initResponse, new WideCallback() {

@Override

public void onCompleted(WKYCResponse response) {

}

@Override

public void onInterrupted(WKYCResponse response) {

}

@Override

public void onCancel(WKYCResponse response) {

}

});

The transaction result contains a result code that indicates the transaction flow status. If the end-user has completed the flow, the onCompleted method is invoked, where the transaction status needs to be synced with your (merchant) server and a double check needs to be started. Then your (merchant) server needs to call the checkResult API to get transaction details and return them to your (merchant) application.

If the end-user has not completed the flow or client or server-side runtime error happens, the onInterrupted method is invoked, where related process logic needs to be implemented according to your business requirements.

And if the end-user canceled the process using the title back button or native back on their device, the onCancel method is invoked, where related process logic needs to be implemented according to your business requirements.