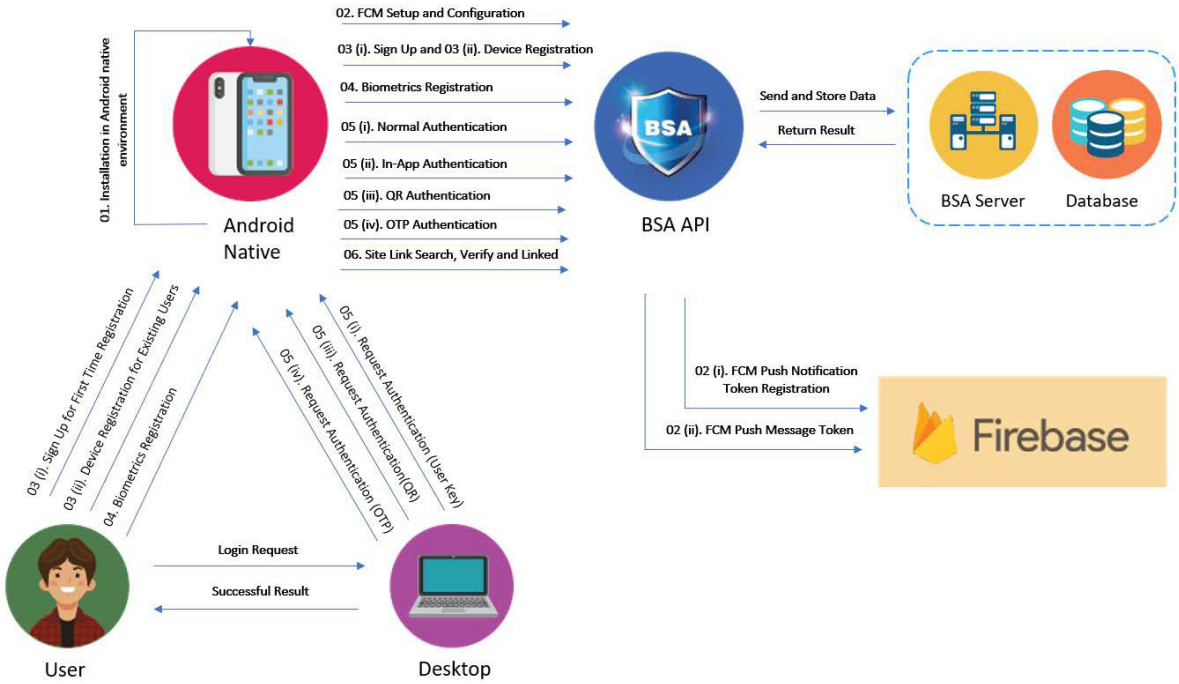
Getting Started

The BSA SDK, referred to as the 'Android SDK' henceforth, equips developers with the necessary tools to integrate BSA authentication into their Android applications. This document outlines the process of developing an Android Native application while making the most effective use of the Android SDK for seamless BSA authentication implementation.

Below is the flow that we are going to show:

1. Prerequisites
2. Installation
3. FCM Setup and Configuration
4. User Registration
5. Device Re-registration
6. Authentication
7. User Information Modification
8. Other APIs (Need for Access Token)
9. Other APIs (No need for Access Token)
10. Error Code



Note that this Android Native sample is written in Kotlin. You may use Java to develop this app.

Prerequisites

Before proceed to the SDK integration, it is advisable to follow the following prerequisites below

Android Studio Version

Minimum	Target
Android 4.4 KitKat	Android 13

Gradle

Minimum	Target
Version 6.0.0	Version 8.1.2

Target Android Version

Minimum	Target
Android 6.0 Marshmallow (API level 23)	Android 13 (API level 33)

Installation

i.Set Gradle

Declare the Android SDK repository in the **build.gradle** (project) to implement Android SDK.

```
maven {
    url "https://nexus.fnsvalue.co.kr/repository/maven-releases/"
}
```

ii. Add Modules

Add necessary modules in the **build.gradle**(app).

```
dependencies {
      //Kindly refer to fnsv maven url in item (i) for latest SDK update
      implementation 'com.bsa.sdk:BsaAuthentication:1.0.7@aar'
}
```

Add dependencies to be manually installed in external library required for Android SDK

```
dependencies {
    // retrofit2
    implementation 'com.squareup.okhttp3:logging-interceptor:3.14.9'
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

    implementation 'com.google.code.gson:gson:2.8.6'

    // websocket
    implementation 'com.github.NaikSoftware:StompProtocolAndroid:1.6.4'

    // biometric
    implementation "androidx.biometric:biometric:1.0.1"

    implementation group: 'io.reactivex.rxjava2', name: 'rxjava', version: '2.2.5'
    implementation 'com.warrenstrange:googleauth:1.4.0'
}
```

iii. Permission Configuration

Set the permissions required to use various features through the Android SDK. It can be setup at the **AndroidManifest.xml** as follows,

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sample">

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT"/>
<!-- Android 13 Notification Runtime -->
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"
    android:minSdkVersion="33"/>

    ....

</manifest>
</manifest>
```

iv. JAVA 8 Configuration

Specify Java version in the **build.gradle**(App) file to use Java 8 features.

```
android {
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    ....
}
```

v. Runtime Permission

Android 6.0 Marshmallow (API 23), or higher version requires the runtime permission to use the resources of device. The following are required permissions of the Android SDK to be declared.

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    if (ContextCompat.checkSelfPermission(this,
```

```
android.Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED
    || ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
        ...
    }
}
```

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    if (ContextCompat.checkSelfPermission(this,
android.Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
        ...
    }
}
```

Starting from Android 13 TIRAMISU (API 33) and above, runtime permissions are required to use the device's resources.

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    if (ContextCompat.checkSelfPermission(this,
android.Manifest.permission.POST_NOTIFICATIONS) !=
PackageManager.PERMISSION_GRANTED) {
        ...
    }
}
```

vi. Initialization

For initial setup, the Android SDK must be initialized after the installation. It can be initialized as follows:

```
class GlobalApplication : Application() {
    override fun onCreate() {
        super.onCreate();

        BsaSdk.getInstance().init(
        applicationContext,
        "{Client Key}",
        "{API SERVER URL}"
        )

    }
}
```

In order to fully utilize the Android SDK, the **client application must be registered to generate the client key.**

FCM Setup and Configuration

In most cases, user will requests BSA authentication from a non-mobile platform, such as a web application, it's important to have the ability to receive push notifications on the associated mobile app that has integrated the Android SDK.

This functionality relies on the proper setup of FCM registration and the capability to update the FCM token. This way, the user can seamlessly receive notifications on their mobile device regardless of the platform they initiated the authentication from.

i. FCM Server Key

In order to receive the push notifications, the **Server Key** generated by Firebase Cloud Messaging (FCM) must be sent together with the client issuance request. The FCM **Server Key** can be found in the process below.

1. Create or load the project in the Firebase Console.
2. In top left, click on " ⚙ " icon located beside Project Overview and click **Project Settings**
3. In the tabs, click on **Cloud Messaging** tab.
4. Copy **Server Key** from Cloud Messaging API. If you do not have the server key yet, You can enable Cloud Messaging API and generate the key. More about Cloud Messaging API can be find in the linked documentation.

It is essential to note that the FCM server key is a crucial component of the process. This key plays a vital role in ensuring the smooth registration of clients on the platform.

ii. FCM Push Notification Token Registration

Registering the FCM push token is pivotal to enable effective notification of authentication requests. Utilizing the **registerPushToken()** function from the Android SDK facilitates the process of linking and updating the FCM push token generated on an Android device. This ensures a streamlined mechanism for handling authentication notifications.

The FCM push token may undergo changes due to factors like expiration. To accommodate such scenarios, it's important to update the FCM push token by utilizing the registerPushToken() function. This ensures that the most current and valid token is in use for seamless notification functionality.

Parameter

Key	Value	Description
token	String	FCM Push Notification

Example

```
//FCM Push Notification Token Registered
FirebaseMessaging.getInstance().token.addOnSuccessListener {
                Log.d("ABCDEFG", "FirebaseMessaging token : $it")
                try {
                    BSASdk.getInstance().registerPushToken(it, object :
```

```
                        SdkResponseCallback<TokenResponse> {
                        override fun onSuccess(result: TokenResponse?) {
                            Log.d("ABCDEFG", "registerPushToken onSuccess :
$result")
                        }
                        override fun onFailed(errorResult: ErrorResult?) {
                            Log.d("ABCDEFG", "registerPushToken onFailed :
$errorResult")
                            Log.d("ABCDEFG", "registerPushToken onFailed rtcode :"
+ errorResult!!.errorCode)
                            Log.d("ABCDEFG", "registerPushToken onFailed message:
+" + errorResult!!.errorMessage)
                }
            })
        } catch (exception: Exception) {
            // Handle any exceptions that may occur
        }
}
// FCM Push Notification Token Renewed
class FirebasePushService : FirebaseMessagingService() {
    override fun onMessageReceived(remoteMessage: RemoteMessage) {
        // Handle incoming push notification message
    }
    override fun onNewToken(token: String) {
        super.onNewToken(token)
        BSASdk.getInstance().registerPushToken(token, object :
SdkResponseCallback<ResultResponse> {
            override fun onSuccess(result: ResultResponse?) {
                Log.d("ABCDEFG", "Result code: ${result?.rtCode}")
            }

            override fun onFailed(errorResult: ErrorResult?) {
                Log.d("ABCDEFG", "Error code: ${errorResult?.errorCode}")
            }
        })
    }
}
```

Result Response

When the API call to register the FCM Push token is successful, the response's **rtCode** will be **0**. This indicates that the registration process was completed without any errors.

key	Value	Description
rtCode	0	ReturnMessage
rtMsg	String	Result Message

Error Result

Key	Value	Description

Key	Value	Description
errorCode	Int	Error Code

iii. FCM Push Message Transfer

If requesting an authentication on web using user key, a message is transferred through FCM on mobile application. In case of receiving a push message after an authentication, it proceeds an authentication process by calling **responsePushMessage** of **BSASdk**.

Parameter

Key	Value	Description
remoteMessage	Map<String.String>	Authentication push message

Example

```
class FirebasePushService : FirebaseMessagingService() {
    override fun onMessageReceived(remoteMessage: RemoteMessage) {
        BSASdk.getInstance().responsePushMessage(remoteMessage.data)
    }

    override fun onNewToken(token: String) {
        // Leave the method empty or add your desired implementation here
    }
}
```

User Registration

Overview

This guide explains the registration and integration method of the BSA via the Android SDK.

Feature Description

Provides the feature to register and integrate with BSA using member information.

i. User ID Duplication Check

Before proceeding with BSA registration, check if there is a duplicate user ID.
Use `BsaSdk`'s `isDuplicateUserKey()` to request the API.

Example

```
BsaSdk.getInstance().sdkService.isDuplicateUserKey(userKey, object:
SdkResponseCallback<CheckDuplicateUserKeyResponse> {
    override fun onSuccess(result: CheckDuplicateUserKeyResponse) {
        // Code to run if user ID is not duplicated
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        // Code to run if user ID is duplicated or connection failed
        ...
    }
})
```

Parameter

Name	Value	Description
userKey	String	User key to check for duplication

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

ResultCode

rtCode	Description
0	Success
2009	In case of an already registered user ID

ii. Email or Phone Number Duplication Check

Before proceeding with BSA registration, check if there is a duplicate email or phone number.
Use `BsaSdk`'s `isDuplicatedEmailOrPhoneNumber()` to request the API.

Example

```
val params : HashMap<String, String> = HashMap()
params["verifyType"] = SdkConstant.OtpType.EMAIL.value
params["verifyData"] = "{email}"
// or for SMS
// params["verifyType"] = SdkConstant.OtpType.SMS.value
// params["verifyData"] = "phoneNumber"

BsaSdk.getInstance().sdkService.isDuplicatedEmailOrPhoneNumber(params, object:
SdkResponseCallback<CheckDuplicateEmailOrPhoneNumberReponse> {
    override fun onSuccess(result: CheckDuplicateEmailOrPhoneNumberReponse?) {
        // Code to run if not duplicated
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        // Code to run if duplicated or connection failed
        ...
    }
})
```

Parameter

Name	Value	Description
verifyType	String	Data type to check for duplication (Use SdkConstant.OtpType)
verifyData	String	Data to check for duplication

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

ResultCode

rtCode	Description
0	Success
2019	In case of already registered data

iii. Email Verification Request

To register with BSA, you need to verify your email. Request email verification by using `BsaSdk`'s `sendOtpByEmail()` API. If the request is successful, a registration verification code will be sent to the input email.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{CLIENT_KEY}"
params["email"] = "{email}"

BsaSdk.getInstance().sdkService.sendOtpByEmail(params, object:
SdkResponseCallback<SendOtpResponse> {
    override fun onSuccess(result: SendOtpResponse?) {
        // Code to run after successful request
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
clientKey	String	Unique key of the client
email	String	User's email
phoneNum	String	User's phone number

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

iv. Email Verification Confirmation

Use the `seq` result value from `sendOtpByEmail()`, the registration verification code shown in the email, and input the email to confirm email verification. Confirm email verification by using `BsaSdk`'s `verifyOtpByEmail()` API. If the registration verification code is valid, the `token` for registration will be provided.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{CLIENT_KEY}"
```

```
params["email"] = "{email}"
params["authNum"] = "{authNumber}"

BsaSdk.getInstance().sdkService.verifyOtpByEmail(params, object: Sdk

ResponseCallback<VerifyOtpResponse> {
    override fun onSuccess(result: VerifyOtpResponse?) {
        // Code to run after email verification
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
clientKey	String	Unique key of the client
email	String	User's email
authNum	String	OTP number entered by the user

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{data}	class	
data.result	Boolean	Verification result
data.disposeToken	String	Verification token

v. SMS Verification Request

To register with BSA, SMS verification is required. Request SMS verification by using `BsaSdk`'s `sendOtpBySms()` API. If the request is successful, a registration verification code will be sent via SMS.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{CLIENT_KEY}"
params["phoneNum"] = "{phoneNumber}"

BsaSdk.getInstance().sdkService.sendOtpBySms(params, object:
SdkResponseCallback<SendOtpResponse> {
    override fun onSuccess(result: SendOtpResponse?) {
```

```
        //
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
clientKey	String	Unique key of the client
phoneNum	String	User's phone number

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

vi. SMS Verification Confirmation

Use the `seq` result value from `sendOtpByEmail()`, the registration verification code shown in the SMS, and input the SMS to confirm SMS verification. Confirm SMS verification by using `BsaSdk`'s `verifyOtpBySms()` API. If the registration verification code is valid, the `token` for registration will be provided.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{CLIENT_KEY}"
params["phoneNum"] = "{phoneNumber}"
params["authNum"] = "{authNumber}"

BsaSdk.getInstance().sdkService.verifyOtpBySms(params, object:
SdkResponseCallback<VerifyOtpResponse> {
    override fun onSuccess(result: VerifyOtpResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description

Name	Value	Description
clientKey	String	Unique key of the client
phoneNum	String	User's phone number
authNum	String	OTP number entered by the user

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{data}	class	
data.result	Boolean	Verification result
data.disposeToken	String	Verification token

vii. Biometric Authentication Registration

This is a mandatory process during user registration. It is a method to register biometric authentication information.

Example

```
BsaSdk.getInstance().sdkService.registerBiometric(fragmentActivity, object:
SdkResponseCallback<AuthBiometricResponse> {
    override fun onSuccess(authBiometric: AuthBiometricResponse?) {
        // Code to execute after registration
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

- none

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

viii. Registration

To register with BSA, request the API using `BsaSdk`'s `registerUser()`.
You need the `token` received after confirming email or SMS verification in addition to the basic information.

Example

```
val params: MutableMap<String, Any> = HashMap()
params["userKey"] = "{userKey}"
params["name"] = "{name}"
params["phoneNum"] = "{phoneNum}"
params["email"] = "{email}"
params["authType"] = "{authType}"
params["agreeGccs"] = true
params["agreePerson"] = true
params["agreeDevice"] = true
params["disposeToken"] = "{disposeToken}"

FirebaseMessaging.getInstance().token.addOnSuccessListener { token ->
    params["token"] = token
}

BsaSdk.getInstance().sdkService.registerUser(map, object:
SdkResponseCallback<RegisterUserResponse> {
    override fun onSuccess(result: RegisterUserResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
userKey	String	User ID
name	String	User name
phoneNum	String	User's phone number
email	String	User's email
authType	String	Authentication type
agreeGccs	Boolean	Agreement to terms of use
agreePerson	Boolean	Agreement to personal information processing policy
agreeDevice	Boolean	Agreement to device information use
disposeToken	String	Token received after email or SMS verification
token	String	FCM token

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

Device Re-registration

This guide explains how to deregister and re-register a device with the BSA via the Android SDK.

If a user who was using BSA changes their mobile device, the device re-registration feature allows them to use BSA authentication as before by verifying member information and re-registering the device.
First, the membership status is checked, and then an OTP code is sent via email for member verification, followed by re-registration.

First, the membership status is checked, and then an OTP code is sent via email for member verification, followed by re-registration.

i. User Check and OTP Dispatch

Request for user check and OTP dispatch. Use `BsaSdk`'s `sendOtpByRegisterDevice()` to request the API. If the entered user information is correct, an OTP code will be sent to the email.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{clientKey}"
params["userKey"] = "{userKey}"
params["name"] = "{name}"
params["verifyType"] = SdkConstant.OtpType.EMAIL.value
params["verifyData"] = "{email}"
// or for SMS
// params["verifyType"] = SdkConstant.OtpType.SMS.value
// params["verifyData"] = "{phoneNum}"

BsaSdk.getInstance().sdkService.sendOtpByRegisterDevice(params, object:
SdkResponseCallback<SendOtpResponse> {
    override fun onSuccess(result: SendOtpResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Key	Value	Description
clientKey	String	Unique key of the client
userKey	String	User ID
name	String	User name
verifyType	String	Verification type

Key	Value	Description
verifyData	String	Verification data (email or phone number)

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{data}	class	
data.result	Boolean	Verification result
data.authType	Int	Authentication type

ii. Email Verification Confirmation

Use the `seq` result value from `sendOtpByEmail()`, the registration verification code displayed in the email, and input the email to confirm email verification. To confirm email verification, request the API using `BsaSdk`'s `verifyOtpByEmail()`. If the registration verification code is valid, the `token` for registration will be provided.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{CLIENT_KEY}"
params["email"] = "{email}"
params["authNum"] = "{authNumber}"

BsaSdk.getInstance().sdkService.verifyOtpByEmail(params, object:
SdkResponseCallback<VerifyOtpResponse> {
    override fun onSuccess(result: VerifyOtpResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
clientKey	String	Unique key of the client
email	String	User's email
authNum	String	OTP number entered by the user

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{data}	class	
data.result	Boolean	Verification result
data.disposeToken	String	Verification token

iii. SMS Verification Confirmation

Use the `seq` result value from `sendOtpBySms()`, the registration verification code displayed in the email, and input the email to confirm email verification. To confirm email verification, request the API using `BsaSdk`'s `verifyOtpBySms()`. If the registration verification code is valid, the `token` for registration will be provided.

Example

```
val params : HashMap<String, Any> = HashMap()
params["clientKey"] = "{CLIENT_KEY}"
params["phoneNum"] = "{phoneNumber}"
params["authNum"] = "{authNumber}"

BsaSdk.getInstance().sdkService.verifyOtpBySms(params, object:
SdkResponseCallback<VerifyOtpResponse> {
    override fun onSuccess(result: VerifyOtpResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
clientKey	String	Unique key of the client
phoneNum	String	User's phone number
authNum	String	OTP number entered by the user

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

Name	Type	Description
{data}	class	
data.result	Boolean	Verification result
data.disposeToken	String	Verification token

iv. Local Authentication

This is a mandatory step for authenticating the device using the previously registered local authentication method.

This is a method for registering biometric authentication information.

Example

```
BsaSdk.getInstance().sdkService.registerBiometric(fragmentActivity, object:
SdkResponseCallback<AuthBiometricResponse> {
    override fun onSuccess(authBiometric: AuthBiometricResponse?) {
        // Code to execute after registration
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

This is a method for device authentication or registering device authentication information(PIN, pattern).

Example

```
BsaSdk.getInstance().sdkService.authDeviceCredential(fragmentActivity, object :
SdkResponseCallback<AuthBiometricResponse> {
    override fun onSuccess(authBiometric: AuthBiometricResponse?) {
        // Code to execute after successful password/pattern authentication
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

v. Device Re-registration

Request the device re-registration API using `BsaSdk`'s `reRegisterUserDevice()`.
You need the token verified through `verifyOtpByEmail()`, and after the re-registration process is completed, you can use BSA authentication as before.

Example

```
val params: MutableMap<String, Any> = HashMap()
params["userKey"] = "{userKey}"
params["name"] = "{name}"
params["phoneNum"] = "{phoneNum}"
params["email"] = "{email}"
params["authType"] = "{authType}"
params["otpType"] = "{email}" // or for SMS it's "{sms}"
params["disposeToken"] = "{disposeToken}"

FirebaseMessaging.getInstance().token.addOnSuccessListener { token ->
    params["token"] = token
}

BsaSdk.getInstance().sdkService.reRegisterUserDevice(map, object:
SdkResponseCallback<ReRegisterDeviceResponse> {
    override fun onSuccess(result: SendOtpResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Value	Description
userKey	String	User ID
name	String	User name
phoneNum	String	User's phone number
email	String	User's email
authType	String	Authentication type
otpType	String	OTP verification type
disposeToken	String	Token received after email or SMS verification
token	String	FCM token

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

If the device re-registration API call is successful, you will receive a `rtCode` of 0.

Authentication

With BSA authentication, user can be verified without any password.

In the process of authentication request to node verification, the token will be provided if the user has passed without any trouble.

The token is used for API features such as checking authentication history.

i. Normal Authentication

This is a method for processing authentication requests received from external sources.

Example

```
BsaSdk.getInstance().sdkService.normalAuthenticator(userKey, isAuth,
fragmentActivity, object: SdkAuthResponseCallback<AuthCompleteResponse> callback {
    override fun onSuccess(result: AuthCompleteResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
userKey	String	User ID
isAuth	Boolean	Success of the previous authentication step ('Authentication Start' or 'Biometric Authentication')
fragmentActivity	FragmentActivity	An activity object to display local authentication

Response

Name	Type	Description
rtCode	Int	Result Code
rtMsg	String	Result Message

ii. In-App Authentication

This is a method for handling authentication requests generated within the app.

Example

```
BsaSdk.getInstance().sdkService.appAuthenticator(userKey, isAuth,
fragmentActivity, object: SdkAuthResponseCallback<AuthResultResponse> callback) {
    override fun onSuccess(result: AuthResultResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
userKey	String	User ID
isAuth	Boolean	Success of the previous authentication step ('Authentication Start' or 'Biometric Authentication')
fragmentActivity	FragmentActivity	An activity object to display local authentication

Response

Name	Type	Description
rtCode	Int	Result Code
rtMsg	String	Result Message

iii. QR Authentication

This is a method used for processing authentication via QR codes.

Example

```
BsaSdk.getInstance().sdkService.qrAuthenticator(userKey, qrId, isAuth,
fragmentActivity, object: SdkAuthResponseCallback<AuthCompleteResponse> callback {
    override fun onSuccess(result: AuthCompleteResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
userKey	String	User ID

Name	Type	Description
qrId	String	QR ID extracted through QR library
isAuth	Boolean	Success of the previous authentication step ('Authentication Start' or 'Biometric Authentication')
fragmentActivity	FragmentActivity	An activity object to display local authentication

Response

Name	Type	Description
rtCode	Int	Result Code
rtMsg	String	Result Message

iv. OTP View Authentication

This is a method used for processing authentication with OTP (One-Time Password) numbers.

Example

```
BsaSdk.getInstance().sdkService.otpViewAuthenticator(userKey, isAuth,
fragmentActivity, object: SdkAuthResponseCallback<AuthResultResponse> callback) {
    override fun onSuccess(result: AuthResultResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
userKey	String	User ID
isAuth	Boolean	Success of the previous authentication step ('Authentication Start' or 'Biometric Authentication')
fragmentActivity	FragmentActivity	An activity object to display local authentication

Response

Name	Type	Description
rtCode	Int	Result Code
rtMsg	String	Result Message

ErrorResult

It returns an error code when it fails to call API for the authentication cancel.

Key	Value	Description
errorCode	Int	Error Code
errorMessage	String	Error Message

v. Authentication Cancellation

Request authentication cancellation using BsaSdk's **cancelAuth()**.

Example

```
BsaSdk.getInstance().sdkService.cancelAuth(object:
SdkResponseCallback<AuthCancelResponse> {
    override fun onSuccess(authCancel: AuthCancelResponse?) {
        // Code to be executed after 'Authentication Cancellation' success
        ...
    }
    override fun onFailed(authFailed: ErrorResult?) {
        ...
    }
})
```

Parameter

none

AuthCancelResponse

Key	Value	Description
rtCode	Int	Return code
rtMsg	String	Return message

vi. Checking Existing Authentication Requests

If you want to check if there's currently an ongoing authentication process, you can request information using BsaSdk's **existAuth()**.

Example

```
BsaSdk.getInstance().sdkService.existAuth(userKey, object:
SdkResponseCallback<AuthExistResponse> {
    override fun onSuccess(authExist: AuthExistResponse?) {
        // Code to be executed if there's currently an ongoing authentication
```

```
        ...
    }
    override fun onFailed(authFailed: ErrorResult?) {
        ...
    }
})
```

Parameter

Key	Value	Description
userKey	String	User ID

AuthExistResponse

Key	Value	Description
rtCode	Int	Return code
rtMsg	String	Return message
data.isExist	Boolean	Whether authentication is in progress or not
data.clientKey	String	Unique client key
data.siteUrl	String	Site URL where authentication is in progress
data.timeout	String	Expiry time
data.clientName	String	Client name

vii. Retrieving OTP Number

Retrieve the OTP number for BSA authentication by requesting the API using BsaSdk's **getAuthOtpCode()**.

Example

```
BsaSdk.getInstance().sdkService.getAuthOtpCode(object:
SdkResponseCallback<OtpGenerateResponse> {
    override fun onSuccess(result: OtpGenerateResponse?) {
        // Use the received OTP code
        val otpCode = result.data
        ...
    }
    override fun onFailed(authFailed: ErrorResult?) {
        ...
    }
})
```

Parameter

- none

OtpGenerateResponse

When it succeeds to call API for the authentication result, it returns 0 for rtCode and the ongoing authentication process is cancelled.

Key	Value	Description
rtCode	Int	Return code
rtMsg	String	Return message
data	String	OTP number

viii. OTP Number Cancellation Request

Cancel (expire) the BSA authentication OTP number by requesting the API using BsaSdk's **cancelOtp()**.

Example

```
BsaSdk.getInstance().sdkService.cancelOtp(otpCode, object:
SdkResponseCallback<OtpCancelResponse> {
    override fun onSuccess(result: OtpCancelResponse?) {
        ...
    }
    override fun onFailed(authFailed: ErrorResult?) {
        ...
    }
})
```

Parameter

Key	Value	Description
otpCode	String	OTP number

OtpCancelResponse

Key	Value	Description
rtCode	Int	Return code
rtMsg	String	Return message

User Information Modification

i. Change Local Authentication Type

Use `BsaSdk`'s `resetBiometricChange()` to reset the biometric authentication information in the app.

Example

```
BsaSdk.getInstance().sdkService.resetBiometricChange(fragmentActivity, object:
SdkResponseCallback<AuthBiometricResponse> {
    override fun onSuccess(authBiometric: AuthBiometricResponse?) {
        // Code to execute after reset
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

- none

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

ii. Unregister Device

After deregistration is completed, the device can be used again by re-registering it using the registration information.

Example

```
BsaSdk.getInstance().sdkService.unRegisterDevice(userKey, object:
SdkResponseCallback<UnRegisterDeviceResponse> {
    override fun onSuccess(response: UnRegisterDeviceResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Key	Value	Description
userKey	String	User ID

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

iii. Delete Account

This is a method for requesting membership withdrawal.

Example

```
BsaSdk.getInstance().sdkService.deleteUser(object:
SdkResponseCallback<DeleteUserResponse> {
    override fun onSuccess(response: DeleteUserResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }  )
```

Parameter

- none

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

Other APIs (Need for Access Token)

The following APIs require a valid Access Token obtained through authentication to be invoked successfully.

i. Authentication History Query

Example

```
BsaSdk.getInstance().sdkService.getAuthHistory(page, size, object: object:
SdkResponseCallback<AuthHistoryResponse> {
    override fun onSuccess(response: AuthHistoryResponse?) {
        // Code to execute after querying
        ...
    }

    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
page	Int	Page number
size	Int	Size per page

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{page}	class	
page.first	boolean	Check if the current data is on the first page.
page.last	boolean	Check if the current data is on the last page.
page.currentPage	Int	Current page number.
page.pageSize	Int	Number of data entries per page.
page.totalElements	Int	Total number of data entries.
page.totalPages	Int	Total number of pages calculated based on (pageSize).
{data}	class	
data.seq	Int	Serial number

Name	Type	Description
data.clientKey	String	Unique client key
data.clientName	String	Client name
data.status	String	status code
data.platform	String	connected OS Informaion
data.regDt	String	Registration date

ii. Linked Site List Query

Through `BsaSdk`'s `getMyLinkSite()`, you can request the API to query the list of linked sites.

Example

```
BsaSdk.getInstance().sdkService.getMyLinkSite(object:
SdkResponseCallback<SiteLinkInfoResponse> {
    override fun onSuccess(siteLinkInfo: SiteLinkInfoResponse?) {
        // Code to execute after querying
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

- None

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
`{data}`	class	
data.seq	Int	Serial number
data.clientKey	String	Unique client key
data.clientName	String	Client name
data.userStatus	String	User status code
data.clientExplain	String	Client description
data.siteUrl	String	Site URL

Name	Type	Description
data.interlock	Boolean	Connection status
data.verifyType	String	Verification type

iii. Search Site List Query

Through `BsaSdk`'s `getSearchLinkSite()`, you can request the API to search for the list of sites to link.

Example

```
BsaSdk.getInstance().sdkService.getSearchLinkSite(clientName, object:
SdkResponseCallback<SiteLinkInfoResponse> {
    override fun onSuccess(siteLinkInfo: SiteLinkInfoResponse?) {
        // Code to execute after successful search
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
clientName	String	Client name

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
`{data}`	class	
data.seq	Int	Serial number
data.clientKey	String	Unique client key
data.clientName	String	Client name
data.userStatus	String	User status code
data.clientExplain	String	Client description
data.siteUrl	String	Site URL
data.interlock	Boolean	Connection status
data.verifyType	String	Verification type

iv. Site Account Authentication

Through `BsaSdk`'s `verifyLinkSite()`, you can verify the site account to link with the API.

Example

```
BsaSdk.getInstance().sdkService.verifyLinkSite(clientKey, id, password, object:
SdkResponseCallback<SiteLinkVerifyResponse> {
    override fun onSuccess(siteLinkInfo: SiteLinkVerifyResponse?) {
        // Code to execute after successful verification
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
clientKey	String	Unique client key
id	String	ID registered on the client site
pw	String	Password of the account registered on the client site

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
token	String	Verification token for site link

v. Site Linking Request

Through `BsaSdk`'s `linkSite()`, you can link a site that uses BSA authentication with the API.

Example

```
BsaSdk.getInstance().sdkService.linkSite(clientKey, siteToken, object:
SdkResponseCallback<SiteLinkResponse> {
    override fun onSuccess(siteLinkInfo: SiteLinkResponse?) {
        // Code to execute after successful addition
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
```

```
    }
})
```

Parameter

Name	Type	Description
clientKey	String	Unique client key
siteToken	String	Token received after site verification

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
data	Boolean	Link result

vi. Site Unlinking Request

Through `BsaSdk`'s `unlinkSite()`, you can request the API to unlink the linked site.

Example

```
BsaSdk.getInstance().sdkService.unlinkSite(clientKey, object:
SdkResponseCallback<SiteLinkResponse> {
    override fun onSuccess(siteLinkInfo: SiteLinkResponse?) {
        // Code to execute after successful unlinking
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
});
```

Parameter

Name	Type	Description
clientKey	String	Unique client key

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message

Name	Type	Description
data	Boolean	Link result

Other APIs (No need for Access Token)

i. Terms and Conditions

Request the Terms of Service API using `BsaSdk`'s `getAgreements()`.

Example

```
BsaSdk.getInstance().sdkService.getAgreements(clientKey, lang, object:
SdkResponseCallback<AgreementListResponse> {
    override fun onSuccess(response: AgreementListResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Key	Value	Description
clientKey	String	Client key
lang	String	Language

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{data}	class	
seq	Int	Sequence number
title	String	Title
type	String	Type of agreement
lang	String	Language
useStatus	String	Usage status
regUserKey	String	Registered user ID
regDt	String	Registration date
clientKey	String	Unique client key
clientName	String	Client name

Name	Type	Description
customizable	String	Customization applicability

ii. Detailed Terms and Conditions

Request the API for the details of the Terms of Service using `BsaSdk`'s `getAgreementDetail()`.

Example

```
BsaSdk.getInstance().sdkService.getAgreementDetail(clientKey, seq, object:
SdkResponseCallback<AgreementDetailResponse> {
    override fun onSuccess(response: AgreementDetailResponse?) {
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Key	Value	Description
clientKey	String	Client key
seq	Int	Terms of Service sequence number

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
{data}	class	
seq	Int	Sequence number
title	String	Title
content	String	Content
type	String	Type of agreement
lang	String	Language
useStatus	String	Usage status
regUserKey	String	Registered user ID
regDt	String	Registration date

Name	Type	Description
clientKey	String	Unique client key
clientName	String	Client name
regUserName	String	Registered user's name
subCltBhv	String	Settings for sub-client independence

iii. Notice

Through `BsaSdk`'s `getNotificationDetail()`, you can request the API to query the notifications.

Example

```
BsaSdk.getInstance().sdkService.getNotificationDetail(page, size, object:
SdkResponseCallback<NotificationResponse> {
    override fun onSuccess(notification: NotificationResponse?) {
        // Code to execute after querying
        ...
    }
    override fun onFailed(errorResult: ErrorResult?) {
        ...
    }
})
```

Parameter

Name	Type	Description
page	Int	Page number
size	Int	Size per page
clientKey	String	Unique client key

Response

Name	Type	Description
rtCode	Int	Result code
rtMsg	String	Result message
`{data}`	class	
data.seq	Int	Serial number
data.title	String	Title
data.content	String	Content
data.patchType	String	OS type

Name	Type	Description
data.version	String	Version information
data.regUserName	String	Registered user name
data.deployDt	String	Deployment date
data.regDt	String	Registration date
{page}	class	
page.first	Boolean	Is first page
page.last	Boolean	Is last page
page.currentPage	Int	Current page number
page.pageSize	Int	Size per page
page.totalElements	Int	Total number of elements
page.totalPages	Int	Total number of pages

Error Code

i. User Error

Error Code	Description	Solution
2000	Invalid client key	check the client key. When initializing the Android SDK, it verifies whether the client key has been used
2008	Unregistered user	check the GCCS sign in status. To check whether the user has been registered or not use me() from GuardianSDK
5001	Authentication timeout	Make request for Authentication once again because previous authentication is no longer valid
5005	Unauthorized user	Contact the person in charge to solve this matter
5006	Permanently suspended user	Contact the person in charge to solve this matter
5007	Withdraw user	Contact the person in charge to solve this matter

ii. Authentication Error

Error Code	Description	Solution
2004	Channel does not exist	Check the url used for initialization. If it happens constantly. Please inquire the person in charge.
2010	User authentication in-progress	Depending on the circumstances. cancel previous authentication and request for new one
5010	Authentication failure	Contact the person in charge to solve this matter
5011	User Authentication canceled	Make the person in charge to solve this matter
5015	Failed to create channel	It can occur when the parameters are not enough. if it happens constantly, Please inquire the person in charge.
5017	Failed to send push notification	Problems with FCM(Firebase Cloud Messaging) etc. Also check whether updated token is a correct one
5022	Verification failure	Node verification failed. If it happens constantly, please inquire the person in charge

iii. Biometric Authentication Error

Error Code	Description	Solution
9001	Current android version does not support biometric authentication	It can occur when the Android verison is 6.0 or lower. When the biometric authentication is unavailable , it will be switched to PIN or pattern authentication automatically
9003	Device does not support biometric authentication	Biometric module is not supported by the device. When the biometrci authentication is unavailable, it will be switched to PIN or pettern authentication automatically.
9004	Biometric information not found from the device.	There is no biometric information in this device. When the biometric authentciation is unavailable, it will be swtiched to PIN or pattern authentication automaticaaly.
9005	Guardian CCS does not have biometric information	Biometric information was not registered in the Android SDK. Use RegisterBiometric() from GuardianSdk to register the biometric information for authentication
9006	Biometric information has been changed	Biometric Information stored in the device has been changed Use reserBiometricChange() to reset the biometric information for authentication
9007	Biometric information already registerd	This occur when biometric information can't be identified during authentication process. using the right biometric information is recommended
9008	Biometric information does not match	It can occur when biometric information can't be identified during authentication process. Using the right biometric information is recommended
9009	Biometric authentication error	it occured due to unspecific reason. Contact the person in charge to solve this matter

iv. Other Error

Error Code	Description	Solution
10002	SDK error	An Execption has accoured in the SDK. Contact the person in charge to solve this matter
10003	Server eror	An Exception has occured in the API server Contact the person in charge to solve this matter
10004	Server connection error	Impossible to connect the server. Check internet connection and the server address used for initialization