Twigbit Ident SDK

Status	Version
DRAFT	0.1.1 unreleased

NOTICE: This is a confidential document.

Changelog

0.1.1

- [ausweisident] Return Result URL directly, refactor call redirects into optional method in AusweisIdent helper.
- [ausweisident] Server implementation guide.
- [core] Explicitly handle result URL.
- [core] Refactor state callbacks into interface.
- [core] Review inheritance model and draft alternative livecycle-aware architecture that offers more flexibility.

Roadmap/Backlog

- Make inheritance from IdentificationActivity optional by making the `IdentificationManager` livecycle aware.
- Vibrate on NFC message.

The Twigbit Ident SDK is a lightweight convenience layer on top of the AusweisApp2 SDK written in Kotlin. We are aiming to extract and eliminate the recurring code and configuration that every developer faces integrating the SDK.

Features

- Simplify the tedious AusweisApp2 SDK configuration
- Replace the JSON based messaging system by convenient wrapper methods, giving developers to must-have convenience such as code completion
- Lightweight — besides the AusweisApp2 SDK, the only other dependency is Google GSON for JSON parsing
- Drop-in UI — Provide a simple, customizable drop in UI as a quick integration with identification processes
- (coming soon) Capability check- check whether the users device has the required architecture and NFC capabilities
- (coming soon) A custom identification app as a zero-dependency option for the integration
- (uncertain) Provides a fallback to prompt the user to install the official [AusweisApp2] (https://www.ausweisapp.bund.de/) in case of unsupported architecture (see limitations below)

Limitations

- The AusweisApp2 SDK only supports arm64-v8a architecures since version 15.03. Unfortunalety, we are bound to that limitation.

Usage

The usage examples are provided in Kotlin. The integration works in Java analogously.

Download

To get access to the SDK, please get in touch.

Gradle:

```
dependencies {
  implementation 'com.twigbit.identsdk:identsdk:1.0.0'
}
```

Maven:

```
<dependency>
  <groupId>com.twigbit.identsdk</groupId>
  <artifactId>identsdk</artifactId>
  <version>1.0.0</version>
</dependency>
```

Option 1: Identify users with the Drop-In UI (alpha)

To get started quickly and have the SDK take care of the entire identification process for you, you can use the build-in Drop-in UI.

To start an identification process, simply create a `DropInRequest` with your servers tcTokenURL and start the activity for the result.

```
val REQUEST_CODE_IDENTIFICATION = 0

private fun startDropInIdentification(){
    val dropInRequest = DropInRequest("https://...") // your tcToken
Endpoint
    startActivityForResult(dropInRequest.getIntent(this),
REQUEST_CODE_IDENTIFICATION)
}
```

To receive the identification result, you should override your activities `onActivityResult`.

```
    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        if (requestCode == REQUEST_CODE_IDENTIFICATION) {
            if (resultCode == Activity.RESULT_OK) {
                // Success. Update the UI to reflect the successful
identification
                // and fetch the user data from the server where they were
delivered.
                val resultUrl =
data.getParcelableExtra(IdentificationManager.EXTRA_DROPIN_RESULT)
            } else if (resultCode == Activity.RESULT_CANCELED) {
                // The user canceled the identification
            } else {
                // An error occured during the identification
            }
        }
    }
```

Option 2: Implement your own UI

To receive the SDK's identification state callbacks in your activity, implement the `IdentificationManager.Callback` interface and extend the `IdentificationActivty` to bind an `IdentificationManager` instance to your activities lifecycle.

In your activities `onCreate` method, add the callback to the manager and start the identifcation process.

> ***Note:*** *As the callback method might be called from a different thread, be sure to run all UI operations on your UI thread explicitly.*

```
class MainActivity : IdentificationActivity() {

   val identificationCallback = object: IdentificationManager.Callback{
        override fun onCompleted(resultUrl: String) {
            // The identification was complete, display a success message
to the user and fetch the identification result from the server using the
resultUrl
        }

        override fun onRequestAccessRights(accessRights:
ArrayList<String>) {
            // A list of the fields that the sdk is trying to access has
arrived. Display them to the user and await his confirmation.
            // TODO continue with runIdent()
        }

        override fun onCardRecognized(card: IdentificationCard) {
            // A card was attached to the NFC reader
            // TODO @dev implement card model from JSON message params.
        }

        override fun onRequestPin() {
            // The id cards PIN was requested. Display a PIN dialog to the
user.
            // To continue the identification process, call
identificationManager.setPin(pin: String)
        }

        override fun onRequestPuk() {
            // The id cards PUK was requested. Display a PUK dialog to the
user.
            // To continue the identification process, call
identificationManager.setPuk(puk: String)
        }

        override fun onRequestCan() {
            // The id cards CAN was requested. Display a CAN dialog to the
user.
            // To continue the identification process, call
identificationManager.setCan(can: String)
        }

        override fun onError(error: IdentificationError) {
            // An error occured. Display an error/issue dialog to the
user.
        }
    }


    override fun onCreate(savedInstanceState: Bundle?) {
        identificationManager.addCallback(identificationCallback)
    }
}
```

To start the identification process call the `identificationManagers.startIdent` method with your `tcTokenUrl`.

```
identificationManager.startIdent(tcTokenUrl)
```

> ***Note:** We are working on a lifecycle aware alternative to the IdentificationActivity to give you the flexibility to inherit from your own base activity. This requires you to to override the `onNewIntent` method of the activity and pass down intents to the identificationManager:*
>
> ```
> override fun onNewIntent(intent: Intent?) {
>     super.onNewIntent(intent)
>     val tag = intent!!.getParcelableExtra<Tag>fdapter.EXTRA_TAG)
>     if (tag != null) {
>         identificationManager.dispatchTag(tag)
>     }
> }
> ```

Usage with AusweisIdent (alpha)

This SDK provides useful helpers, if you are planing to use AusweisIdent offered by the Bundesdruckerei GmbH and Governikus KG.

In order to use AusweisIdent you need to provide the Ident SDK with a tcTokenURL pointing to an AusweisIdent server.

```
val ausweisIdentTcTokenUrl = AusweisIdentBuilder()
        .scope(AUSWEISIDENT_SCOPE_FAMILYNAMES)
        .scope(AUSWEISIDENT_SCOPE_PLACEOFBIRTH)
        .state("123456")
        .clientId("ABCDEFG")
        .redirectUrl("https://yourserver.com")
        .build()
```

Then, you can start the identification process like described above, passing the `ausweisIdentTcTokenUrl` as a paramter.

```
identificationManager.startIdent(ausweisIdentTcTokenUrl)
```

Get the result

You will get an url from the SDK as described above. Calling this url will result in several redirects with the last redirect pointing to your redirectUrl with the `code` query parameter after a successful identification or an `error` and `error_description` parameter in case of an error. Your server needs this `code` to receive the user info.

```
https://localhost:10443/demo/login/authcode?
code=S6GKv5dJNwy6SXlRrllay6fcaoWeUWjA6ar5gahrGSI823sFa4&state=123456
```

> ***Warning:*** *If you decide to call the url on your own (and not pass it to a browser) you need to make sure to store and send cookies between the redirects.*

> ***Note:*** *We are working on implementing helper methods to simplify this process.*

Server side implementation

1. Use the *code* to obtain an *access token* from the AusweisIdent OAuth2 Token Endpoint.
2. Use the *access token* to get an *user info token* via the OAuth2 User Info Endpoint containing the personal data from the identification document.

Please see the AusweisIdent documentation for further details or check out our server sample (coming soon).

Sample

A working implementation can be found in the `/samples` directory. Please note that you need a test PA to test the identification flow in the reference system.

Copyright