ZenID Android SDK samples

A collection of samples that shows how to use the ZenID Android SDK. The SDK can help you with performing the following operations on documents:

- OCR and data extraction
- verification of authenticity
- real-time face liveness detection

Identity cards, driving licenses and passports from Czechia and Slovakia are supported.

Samples

- **[basic-sample]** - Shows how to run document scanner or face liveness detector.
- **[basic-dagger-sample]** - Extends basic-sample with the well-known dependency injection framework Dagger.
- **[full-dagger-sample]** - Shows how to compose a full flow (scan a document, pass the liveness check and make a selfie).

Instalation

- Put **sdk-core-release.aar** and **sdk-api-zenid-release.aar** in the libs folder of the app
- Edit the build.gradle file of your app and add

```
ext {
    okHttpVersion = '3.13.0'
    retrofitVersion = '2.5.0'
}

dependencies {
    implementation fileTree(include: ['*.aar'], dir: 'libs')
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.fragment:fragment:1.0.0'
    implementation "com.squareup.okhttp3:okhttp:$okHttpVersion"
    implementation "com.squareup.okhttp3:logging-interceptor:$okHttpVersion"
    implementation "com.squareup.retrofit2:retrofit:$retrofitVersion"
    implementation "com.squareup.retrofit2:converter-gson:$retrofitVersion"
    implementation 'com.jakewharton.timber:timber:4.7.1'
    implementation 'pub.devrel:easypermissions:3.0.0'
    implementation 'com.otaliastudios:cameraview:2.1.0'
}
```

- Rebuild the project

Multi-APK split

The C++ code needs to be compiled for each of the CPU architectures (known as "ABIs") present on the Android environment. Currently, the SDK supports the following ABIs:

- `armeabi-v7a`: Version 7 or higher of the ARM processor. Most recent Android phones use this
- `arm64-v8a`: 64-bit ARM processors. Found on new generation devices

The SDK binary contains a copy of the native `.so` file for each of these four platforms. You can considerably reduce the size of your `.apk` by applying APK split by ABI, editing your `build.gradle` as the following:

```
android {

  splits {
    abi {
        enable true
        reset()
        include 'arm64-v8a', 'armeabi-v7a'
        universalApk false
    }
  }
}
```

More information on the Android documentation

Initialization

The ZenID Android SDK is a collection of two modules. The first one is the core module for offline image processing. The second one is an API integration to our backend system.

```
ZenId zenId = new ZenId.Builder()
        .applicationContext(getApplicationContext())
        .build();

// Make the instance globally accessible.
ZenId.setSingletonInstance(zenId);

// This may take a few seconds. Please do it as soon as possible.
zenId.initialize();
```

In order to start integration, you will need to get an **API key** and **URL** of the backend system.

```
ApiConfig apiConfig = new ApiConfig.Builder()
        .baseUrl("http://your.frauds.zenid.cz/api/")
        .apiKey("your_api_key")
        .build();
```

```
OkHttpClient okHttpClient = new OkHttpClient().newBuilder()
        // .addInterceptor() if you want to add a logging interceptor and so on
        .build();
```

```
ApiService apiService = new ApiService.Builder()
        .apiConfig(apiConfig)
        .okHttpClient(okHttpClient)
        .build();
```

Please, check out our samples and you will get the whole picture on how it works.

Document verification flow

To begin with the document picture verifier just start with expected role, page and country.

```
ZenId.get().startDocumentPictureVerifier(MyActivity.this, DocumentRole.ID, DocumentPage
```

Or you can start directly the identity document verifier, similar for driving licenses or passports.

```
ZenId.get().startIdentityDocumentVerifier(MyActivity.this, DocumentPage.FRONT_SIDE, Docu
```

Face liveness flow

To begin with the real-time face liveness check just start the FaceLivenessDetectorActivity.

```
ZenId.get().startFaceLivenessDetector(MyActivity.this);
```

Results

Set the `ZenId.Callback` to handle taken pictures by the core part of the ZenId SDK.

```
ZenId.get().setCallback(new ZenId.Callback() {

    @Override
    public void onDocumentPictureTaken(DocumentCountry documentCountry, DocumentRole doc
        Timber.i("picturePath: %s", documentPicturePath);
    }

    @Override
    public void onSelfiePictureTaken(String selfiePicturePath) {
        Timber.i("picturePath: %s", selfiePicturePath);
    }

    @Override
    public void userExited() {
        Timber.i("User left the sdk flow without completing it");
    }
});
```

To run optical character recognition (OCR) and investigate documents (please follow the link at http://your.frauds.zenid.cz/Sensitivity/Validators to get more details what investigation is about), you need to connect to our backend systems. To do so, you need to use our `ApiService` and and make appropriate calls to upload documents, for instance:

```
apiService.postDocumentPictureSample(documentCountry, documentRole, documentCode, docume

    @Override
    public void onResponse(Call<SampleJson> call, Response<SampleJson> response) {
        String sampleId = response.body().getSampleId();
        Timber.i("sampleId: %s", sampleId);
    }

    @Override
    public void onFailure(Call<SampleJson> call, Throwable t) {
        Timber.e(t);
    }
});
```

To run OCR or investigation on more documents altogether (both sides of ID card and/or driving license and so on), you should keep safe `sampleId` of each POST call and then do:

```
apiService.getInvestigateSamples(sampleIds).enqueue(new Callback<InvestigationResponseJ

    @Override
    public void onResponse(Call<InvestigationResponseJson> call, Response<InvestigationR
        Timber.i("investigationId: %s", response.body().getInvestigationId() );
        InvestigationResponseJson investigationResponse = response.body();
        showMinedData(investigationResponse);
        showValidatorResults(investigationResponse);
    }

    @Override
    public void onFailure(Call<InvestigationResponseJson> call, Throwable t) {
        Timber.e(t);
    }
});
```

You can find out more detail inside the full-dagger-sample.