



ZignSec

ID & BIO VERIFICATION MOBILE SDK

Integration guidelines

ZignSec AB
Version 1.0 | March, 2025
PUBLISHED

ID & Bio Verification Mobile SDK

Integration guidelines

Code:	
Version:	1.0
Date of version:	2025-Mar-06
Created by:	Martin Heikkilä Customer Service Manager
Approved by:	Jason Coombes, Head of Risk and Compliance
Confidentiality level:	Confidential

CONFIDENTIAL

Table of Contents

Table of Contents	2
ZignSec Mobile-SDK API Documentation.....	4
Overview	4
Usage.....	4
Document analysis	4
Biometry verification	5
Document Analysis and Biometry Flows.....	5
Document + biometry analysis	9
Core functionalities.....	11
Environments	11
API health endpoint	11
Authentication	11
The recommended flow	12
Pre-requisites	12
Auto-update documents db.....	12
Initialize DocumentReader.....	12
Initialize FaceSDK	13
Issue access token.....	13
Create (authorization) session.....	13
Start liveness.....	14
Add liveness transaction to the session.....	15
Scan document	15
Finalize package	17
Add (document reader) transaction to the session.....	17
Reprocess.....	17
Analyze results	18
Freeing resources.....	19
More technical information	20
Supplier documentation	20
Online demo services.....	20
Calling API directly	20
UI Customization.....	20
Logging	20
Error handling	20

Samples	20
Change History	21

CONFIDENTIAL

ZignSec Mobile-SDK API Documentation

Overview

This document provides technical details for integrating with the ZignSec ID & Bio verification (Scanning) Mobile-SDK, which utilizes Face SDK and Document Reader SDK for identity document verification and biometric verification.

Usage

General approach: the mobile device is used for data collection, while backend services handle analysis. We recommend “zero-trust” approach.

Document analysis

Document Analysis is designed for identity verification services that specialize in capturing, reading, and digitally verifying passports, ID cards, driver’s licenses, visas, and other identity documents. The solution automatically detects the document type against a database of supported IDs before securely transmitting the captured document to the backend for analysis.

We offer global ID coverage with full data page processing, including MRZ (Machine Readable Zone), Visual Inspection Zones, and Barcode verification. This accelerates customer onboarding by providing seamless and secure ID verification, supporting over 14,000 identity documents from 250 countries and territories.

For a complete list of supported documents, please look into our supplier’s documentation <https://docs.regulaforensics.com/#useful-resources> (at the moment of writing it’s <https://downloads.regulaforensics.com/work/SDK/doc/AllDocs/Documents List.xls>). Please note that due to our release cycle, it may take up to 1 month for the document to be fully supported by our services (we use only stable releases, and we run a set of time-consuming tests before each update testing APIs and UIs, so we update DocReader monthly)

Document processing scenario

Full Processing (FullProcess): A processing scenario designed to extract all available document data, including text fields from the visual zone, MRZ, and barcodes. This mode focuses on maximizing data extraction accuracy and achieving a high conversion rate and performs only basic security checks.

Authentication (FullAuth): A processing scenario that extracts all available document data and also includes document authentication checks. This mode verifies security features such as holograms and other embedded document security elements, ensuring the authenticity of the document in addition to data extraction.

Feature	Full Processing	Authentication
Visual zone OCR	✓	✓
Document type identification	✓	✓
MRZ OCR	✓	✓
Barcode recognition	✓	✓
Document location	✓	✓
Graphics cropping	✓	✓
Checking security features	X	✓
Hologram detection	✓	✓
Multipage processing	✓	✓

It’s recommended to use ‘FullProcess’ on a device (and a ‘Full’ database), and ‘FullProcess’ or ‘FullAuth’ on backend.

Biometry verification

Liveness vs. Face Capture, Active vs. Passive Liveness

Face Capture vs. Liveness Detection

- **Liveness Detection:** Ensures the face is **real and live**, preventing spoofing attacks using AI and biometric analysis.
- **Face Capture:** Simply takes a picture of a face but **does not verify if the person is real** (e.g., can be fooled by a photo).

Active vs. Passive Liveness

- **Active Liveness:** Requires user interaction (e.g., blinking, turning head) for **stronger security**.
- **Passive Liveness:** Works in the background using AI (e.g., detecting skin texture, light reflection) for a **seamless experience**.

Use Cases

Feature	Passive Liveness	Active Liveness	Face Capture
Purpose	AI determines liveness silently	User proves they are live	Capturing a face image (*)
User Interaction	None	Required (e.g. move head)	None
Security Level	Moderate to High	High	Low
User Experience	Seamless	Requires effort	Easy
Best For	Smooth onboarding	High-security scenarios	Biometric matching

(*) we recommend using face capture only if other options do not work for you. It doesn't support "zero-trust to mobile" concept

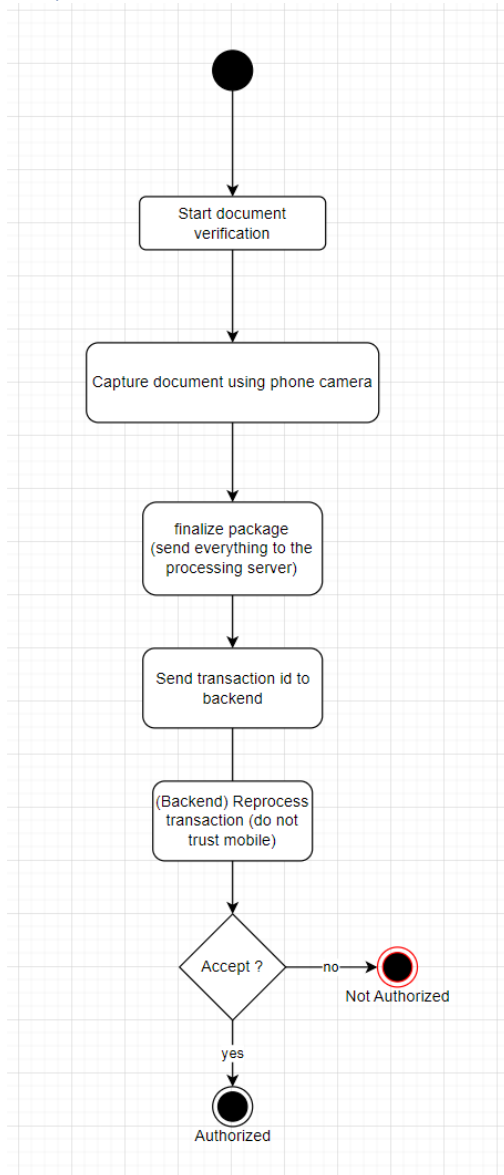
Document Analysis and Biometry Flows

Document analysis is a crucial process for identity authentication, ensuring accuracy and security in digital transactions. The following flows outline different approaches to verifying a document, incorporating various security measures such as **biometric liveness detection**, **RFID processing**, and **retry mechanisms**.

- **Simple Flow** – A basic verification process that captures a document, analyzes it, and makes a final decision.
- **Flow with RFID Check** – Enhances verification by incorporating **RFID chip processing** for documents that support it.
- **Flow with Retry** – Introduces a **retry mechanism**, allowing users to attempt verification again if the initial process fails.
- **Document + Biometry Analysis** – Combines **liveness detection** with document verification, ensuring the person is real before validating their document.

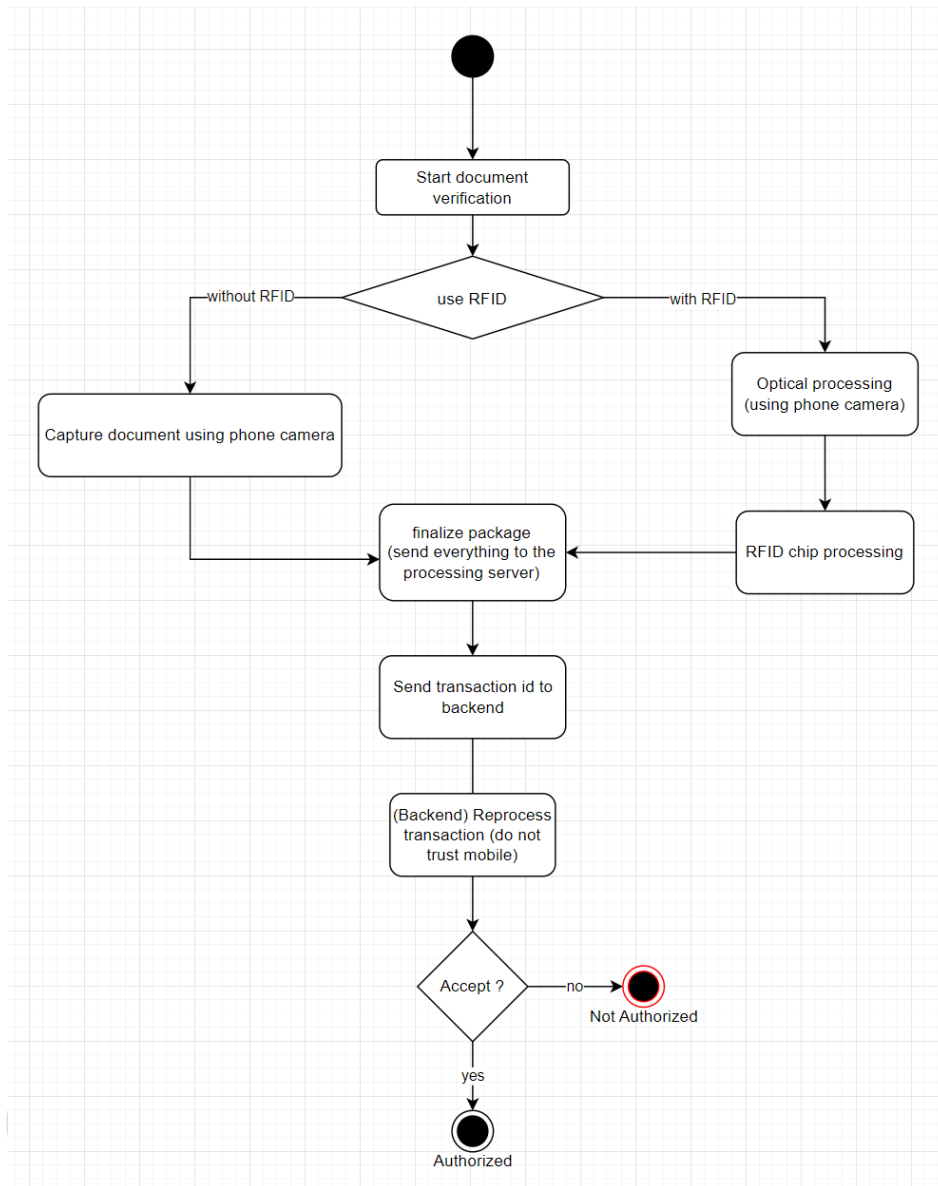
Each of these flows addresses different security and usability needs, optimizing identity verification for various scenarios.

Simple flow



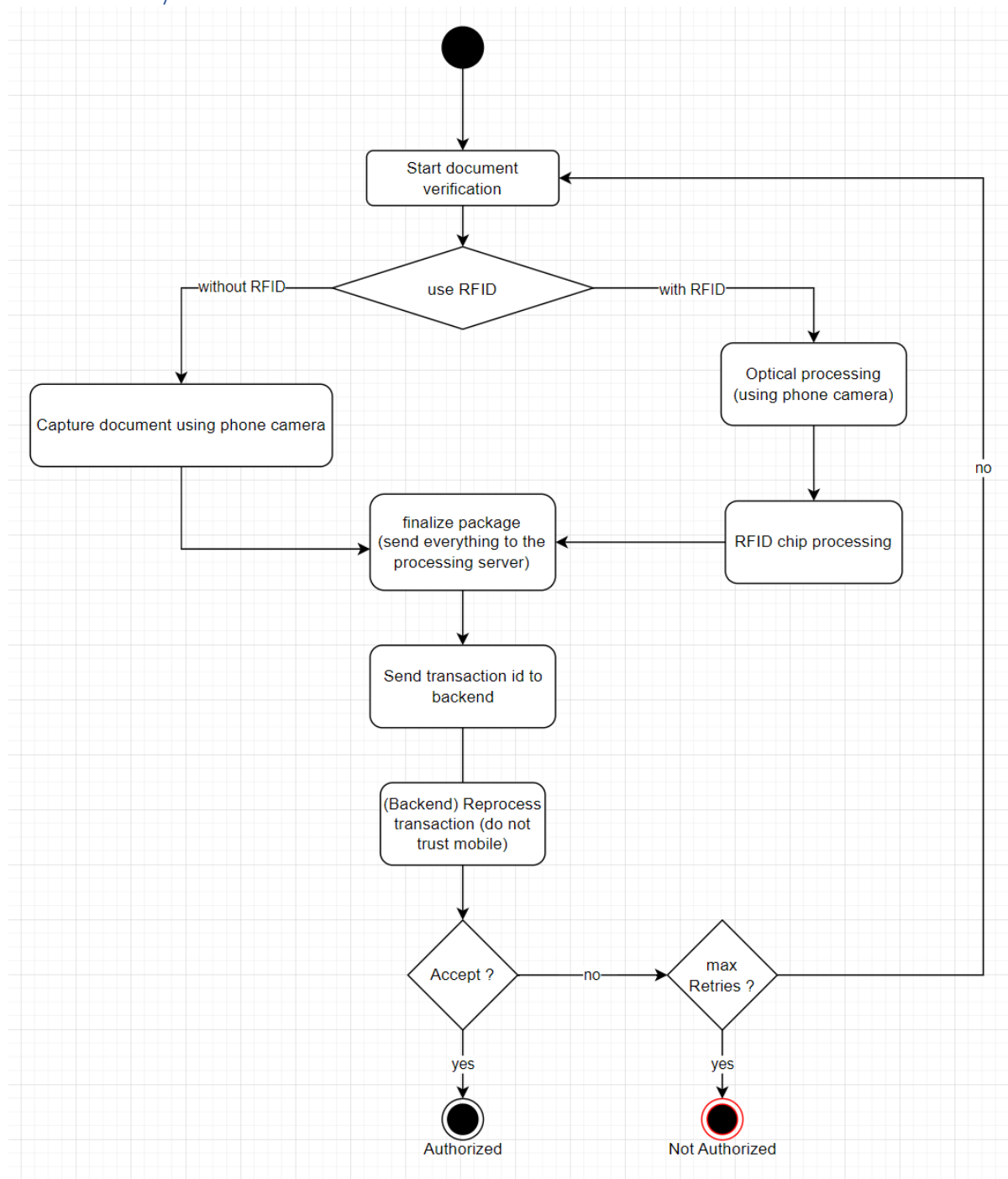
1. **Start Document Verification** – The process is initiated by the user or system.
2. **Capture Document Using Phone Camera** – The user takes a picture of the document using their phone camera. Mobile SDK guides the user through the process
3. **Finalize package** – The captured document and all the related data are encrypted and transmitted to the server, transactionId is returned.
4. **Send transaction id to backend** – the further decision to be done on backend, nothing to be transferred from the device but the transactionId from the previous step
5. **Backend Reprocessing** – Mobile application backend initiates “reprocessing” sending transaction id to the backend.
6. **Decision: Accept?** - Mobile application backend analyzes processing response and makes the final decision to authorize user or not.

Flow with RFID check



1. **Start Verification** – The process is initiated by the user or system.
2. **Decision Point: Use RFID?** – Decide whether to use RFID for document verification.
3. **Without RFID:**
 - a. The user captures the document using a **phone camera**.
4. **With RFID:**
 - a. The document undergoes **optical processing** using the phone camera.
 - b. The **RFID chip is processed** to extract digital information.
5. The remaining steps follow the same process as points 3-6 from the previous flow.

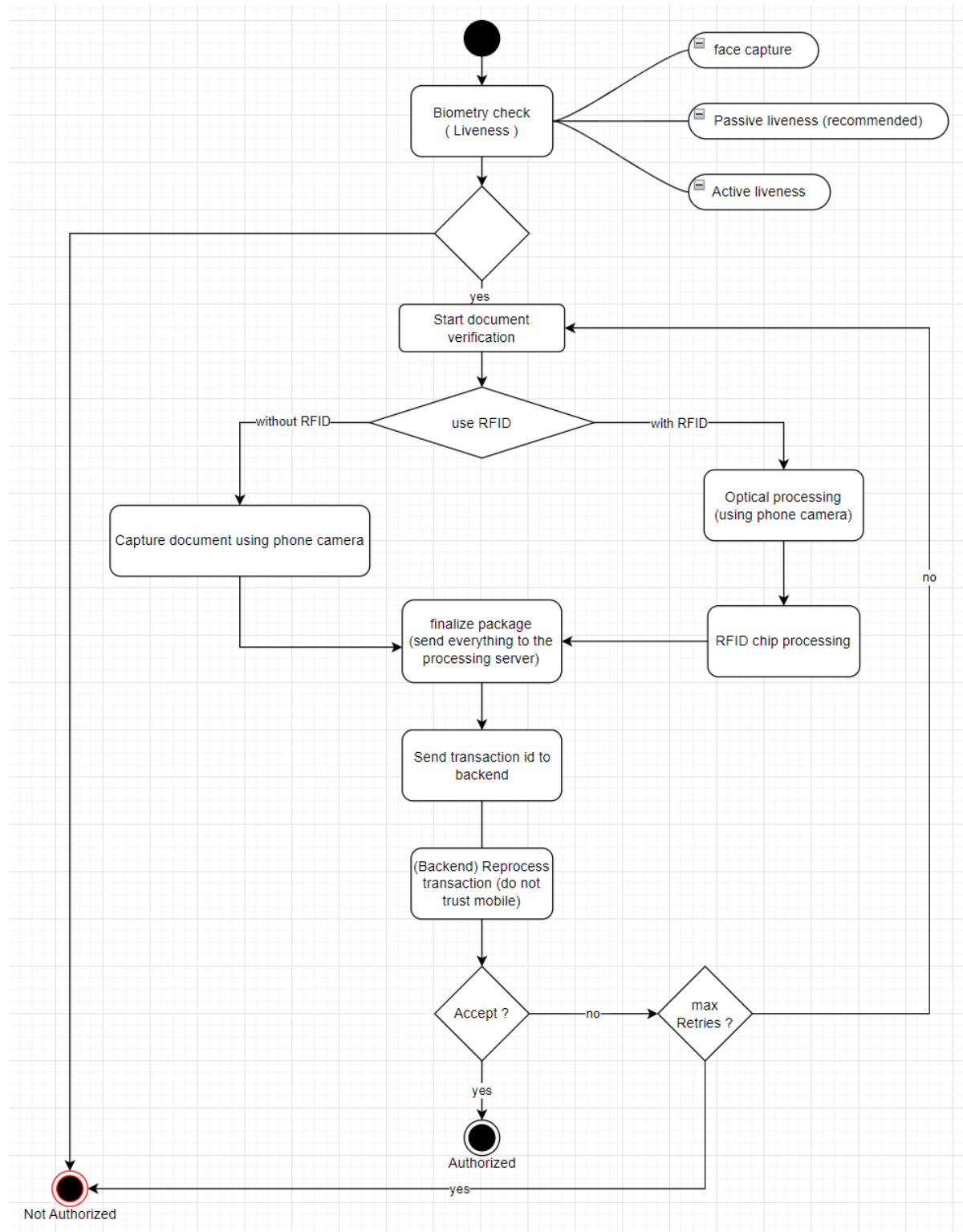
Flow with retry



This flow extends the previous one with retry – user now can retry document verification procedure if not authorized until authorized or max retries limit reached.

Document + biometry analysis

This flowchart outlines a **document verification process combined with biometric (liveness) analysis**, ensuring a secure and reliable identity check.



Steps in the Flow:

1. **Biometry Check (Liveness)** – The process starts with a liveness check, which can be:
 - **Passive Liveness** – the recommended option
 - **Active Liveness** – use if security requirements are more important than the acceptance rate for your business case
 - **Face Capture** – use only if other options do not work for you. Doesn't support "zero-trust to mobile" concept
2. If the liveness check is successful, the system proceeds with the previous flow ([Flow with Retry](#)). Otherwise, the user is not authorized.

CONFIDENTIAL

Core functionalities

Environments

We maintain 2 environments:

- **PROD** <https://gateway.zignsec.com> – for production usage
- **TEST** <https://test-gateway.zignsec.com> – for tests (please contact our support if you need to run a massive/load test)

And there are 3 base URLs for each:

- Face API / Face SDK base URL: [\\$\(baseURL\)/mobilesdk/faceapi](#)
- DocReader base URL: [\\$\(baseURL\)/mobilesdk/docreader](#)
- OpenID configuration: [\\$\(baseURL\)/auth/realms/zignsec/.well-known/openid-configuration](#)

API health endpoint

Both **DocReader** and **Face API** support the /api/ping endpoint to verify that your credentials and our backend are functioning correctly.

For example:

- **TEST** DocReader Ping URL: <https://test-gateway.zignsec.com/mobilesdk/docreader/api/ping>
- **PROD** Face API Ping URL: <https://gateway.zignsec.com/mobilesdk/faceapi/api/ping>

Authentication

OAuth2 Authentication

ZignSec Mobile-SDK uses OAuth2 for secure authentication. Clients must obtain an access token before making API requests using the token endpoint (the endpoint address to be taken from the proper OpenID configuration)

Introspection endpoint (and <https://jwt.io> site) can be used to view token information

Credentials for authentication are supplied by ZignSec support. Contact support@zignsec.com to obtain access credentials.

For detailed API requests and sample usage, refer to the Postman collection:

[ZignSec Mobile-SDK Examples](#)

Authentication headers

2 headers are required to authenticate calls – **Authorization** header should contain Bearer jwt token (received from oauth token endpoint), and **Zs-Product-Key** header should contain product id (subscription key) you receive from our support

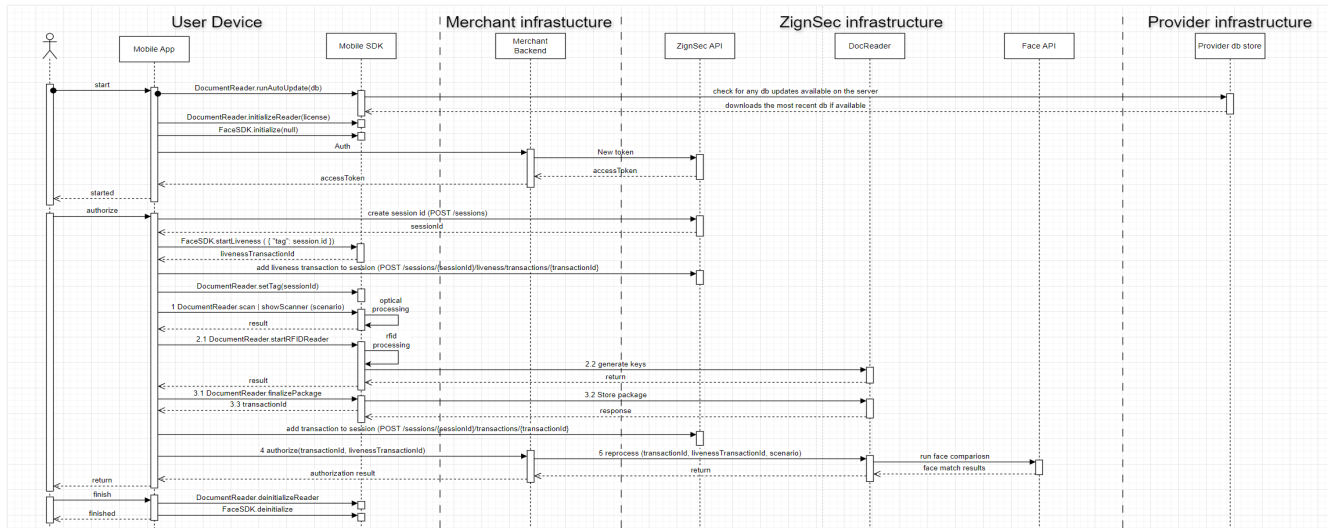
It's recommended to implement a retry policy and retry if 401 or 403 status code returned with a different key. This way we'll be able to rotate keys zero-downtime.

Please also keep in mind that access token is a short-lived token, its lifetime is 20 minutes by default

The recommended flow

We recommend following zero-trust concept and do not rely on any results returned from the mobile sdk, and transaction ids that can be used to initiate the verification on the backend.

Here is the sequence diagram for the most complex scenario ([doc + bio](#)) explained in the first part of this document (rfid part is optional)



Pre-requisites

- DocReader <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/getting-started/installation/>
- Face API <https://docs.regulaforensics.com/develop/face-sdk/mobile/getting-started/installation/#platform-specific-installation-guides>

Auto-update documents db

We recommend using auto-update on app start, Use the “Full” database for the recommended flow. It’s described in suppliers documentation “[Using runAutoUpdate method](#)” – find more examples there

```

DocumentReader.runAutoUpdate("Full",
    (response) => { console.log(response); },
    (error) => { console.error(error); });
  
```

Initialize DocumentReader

You need a valid license on this step (to be provided by our support), see the [documentation](#) for more examples.

```

const secret = '';
DocumentReader.initializeReader({ license: secret },
    (response) => { /* do something useful */ },
    error => console.log(error));
  
```

Initialize FaceSDK

Do not pass any license, we use Core Basic mode (details [here](#))

```
FaceSDK.initialize(null,  
  (response) => { console.log(response); },  
  (error) => { console.error(error); });
```

Issue access token

Please implement a backend service to issue authentication tokens (as client credentials shouldn't be shared with mobile and should only be securely stored on your servers).

We rely on the oauth2 [client credentials](#) flow to issue a short-lived access token that can safely be used from the device (or any other frontend application). The default token lifetime is 20 minutes (please contact our support if you'd like another configuration).

We recommend using [oidc discovery](#) to get a proper url for the token endpoint, for TEST environment run

```
GET https://test-gateway.zignsec.com/auth/realms/zignsec/.well-known/openid-configuration
```

and use its main output to get the access token

```
{  
  "issuer": "https://test-gateway.zignsec.com/auth/realms/zignsec",  
  "token_endpoint": "https://test-gateway.zignsec.com/auth/realms/zignsec/protocol/openid-connect/token",  
  ...  
}
```

And then post to the token endpoint:

```
POST https://test-gateway.zignsec.com/auth/realms/zignsec/protocol/openid-connect/token  
Content-Type: application/x-www-form-urlencoded  
  
grant_type: "client_credentials"  
client_id: "your_client_id"  
client_secret: "your_client_secret"
```

You'll get an access token in response:

```
{  
  "access_token": "eyJ...XYpkg",  
  "expires_in": 1200,  
  "refresh_expires_in": 0,  
  "token_type": "Bearer",  
  "not-before-policy": 0,  
  "scope": "api"  
}
```

At this point you're all set to start authorizing your user.

Create (authorization) session

Use the following request to create a session:

```
POST https://test-gateway.zignsec.com/mobilesdk/sessions  
Headers:  
  Zs-Product-Key: your-subscription-key  
  Authorization: Bearer eyJ...x1w
```

And it returns a session id to be used to identify a current authorization attempt (other fields are informational):

```
{
  "id": "sy89z8wWF0moddTGa0S3dAP8gZm2B2kXl0ie",
  "clientId": "sy89z8wWF0moddTGa0S3dA",
  "clientIdLong": "cf3d2fb3-16cc-4917-a875-d4c66b44b774",
  "uniqueId": "P8gZm2B2kXl0ie"
}
```

This session groups all the necessary verifications (liveness, document, rfid, face match) for a single authorization flow.

Start liveness

Start liveness setting tag to sessionId and specifying the liveness type (see more about liveness types: <https://docs.regulaforensics.com/develop/face-sdk/overview/introduction/#liveness-assessment>)

```
FaceSDK.startLiveness(
  {
    tag: sessionId,
    copyright: false,
    livenessType: passiveLiveness ? LivenessType.PASSIVE :
    LivenessType.ACTIVE,
  },
  (jsonResponse: string) => {
    console.log(jsonResponse);
    const response =
    LivenessResponse.fromJson(JSON.parse(jsonResponse));
    if (response) {
      const livenessTransactionId = response.transactionId;
      console.log(`LivenessTransactionId: ${livenessTransactionId}`);
      if (response.liveness == LivenessStatus.PASSED) {
        console.log("Liveness passed"); }

      if (response.error) {
        console.error(`Liveness error: ${response.error.code} -
        ${response.error.message}`);
        if (response.error.code == LivenessErrorCode.CANCELLED) {
          return; }
        }
      },
      (error) => { console.error(error); }
    );
  }
```

Remember the livenessTransactionId here – we'll use it later to compare the face with the document during server-side reprocessing

Please see online documentation for

- More examples – <https://docs.regulaforensics.com/develop/face-sdk/mobile/feature-usage/liveness/#display-liveness>
- UI customization – <https://docs.regulaforensics.com/develop/face-sdk/mobile/feature-usage/liveness/#ui-customization>
- Error handling – <https://docs.regulaforensics.com/develop/face-sdk/mobile/feature-usage/liveness/#error-handling>

Add liveness transaction to the session

To properly restrict access to the transaction, use the following endpoint to add transaction to the session

```
POST https://test-gateway.zignsec.com/mobilesdk/sessions/{sessionId}/liveness/transactions/{transactionId}
Headers:
  Zs-Product-Key: your-subscription-key
  Authorization: Bearer eyJ...x1w
```

And it returns 204 if all good

Scan document

Start from setting sessionId for DocumentReader

```
DocumentReader.setTag(sessionId, (_) => {}, (_) => {} );
```

Customize widget (see documentation <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/customization/#customize-user-interface>)

```
DocumentReader.setCustomization(
{
  showResultStatusMessages: true,
  showStatusMessages: true,
  showNextPageAnimation: true,
  showHelpAnimation: true,
},
(_) => {},
(_) => {}
);
```

Set default processing parameters (see documentation <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/configuration/#configure-document-processing>)

```
DocumentReader.setProcessParams(
{
  multipageProcessing: true,
  backendProcessingConfig: {
    url: 'https://test-gateway.zignsec.com/mobilesdk/docreader',
    httpHeaders: {
      Authorization: `Bearer ${accessToken}`,
      'Zs-Product-Key': subscriptionKey
    },
  },
  logs: true,
  logLevel: LogLevel.Warning,

  // debugging options, not for production
  debugSaveLogs: true,
  debugSaveCroppedImages: true,
  debugSaveRFIDSession: true,
},
(_) => {},
(_) => {}
);
```


Setup scanner events listener to handle completion

```
const eventManager = new NativeEventEmitter(RNRegulaDocumentReader);
eventManager.addListener("completion", (e) => {
  const data = DocumentReaderCompletion.fromJson(e);
  const action = data?.action

  console.log(`Scanning completion event received: ${action}`, e);

  switch (action) {
    case DocReaderAction.COMPLETE:
      console.log('Scanning completed'); // todo: go to the next step
      break;
    case DocReaderAction.TIMEOUT:
      console.error('Scanning timeout'); // todo: handle timeout
      break;
    case DocReaderAction.ERROR:
      console.error(`Scanning error. Error code:"${e.error?.code}"
Message: "${e.error?.message}"`); // todo: handle error
      break;
    case DocReaderAction.MORE_PAGES_AVAILABLE:
      console.log('More pages available'); // todo: handle
      break;
    case DocReaderAction.CANCEL:
      console.log('Scanning cancelled'); // todo: handle cancellation
      break;
    case DocReaderAction.PROCESS:
    case DocReaderAction.PROCESS_WHITE_FLASHLIGHT:
    case DocReaderAction.NOTIFICATION:
    case DocReaderAction.PROCESS_WHITE_UV_IMAGES:
    case DocReaderAction.PROCESS_IR_FRAME:
    case DocReaderAction.PROCESSING_ON_SERVICE:
    default:
      // nothing to do, wait for the next event
      break;
  }
});
```

And then show scanner

```
const config = new ScannerConfig();
config.scenario = ScenarioIdentifier.SCENARIO_FULL_PROCESS;
config.onlineProcessingConfig = {
  processParams: {},
};
DocumentReader.scan(
  config,
  (_) => {},
  (e) => console.log(e)
);
```

Finalize package

When a user document is scanned, we're sending all the data as a single package to our server for re-processing on backend implementing zero-trust to mobile concept:

```
DocumentReader.finalizePackage(  
  (rawResponse) => {  
    const response = JSON.parse(rawResponse);  
    if (response.error != null) {  
      console.error(response.error);  
      throw new Error("Error sending package for processing");  
    };  
    if (response.action == DocReaderAction.COMPLETE) {  
      const transactionId = response.info.transactionId;  
      console.log('Transaction id:', transactionId);  
    }  
  },  
  (_) => {}  
);
```

Read more: <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/integration/server-side-verification/#step-3-finalize-package>

Add (document reader) transaction to the session

Like the liveness transaction, add the document reader transaction to the session too:

```
POST https://test-gateway.zignsec.com/mobilesdk/sessions/{sessionId}/transactions/{transactionId}  
Headers:  
  Zs-Product-Key: your-subscription-key  
  Authorization: Bearer eyJ...x1w
```

And it returns 204. This information is used to restrict access to the transaction/session data.

Reprocess

Now we're ready to reprocess the package on the backend. Let's setup the client first (see <https://docs.regulaforensics.com/develop/doc-reader-sdk/web-service/development/clients/> to find a proper one for your tech stack)

```
const api = new DocumentReaderApi({  
  basePath: "https://test-gateway.zignsec.com/mobilesdk/docreader",  
  baseOptions: {  
    headers: {  
      Authorization: `Bearer ${accessToken}`,  
      "Zs-Product-Key": subscriptionKey,  
    }  
  },  
});
```

And post a reprocessing request sending transactionId, livenessTransactionId and scenario to use:

```
api.reprocessTransaction(transactionId, {  
  processParam: {  
    scenario: fullAuth ? Scenario.FULL_AUTH : Scenario.FULL_PROCESS,  
    useFaceApi: true,  
    faceApi: {  
      mode: "match",  
      threshold: 75,  
    }  
  }  
});
```

```

        setTimeout: 30000
    },
    dateFormat: "yyyy-MM-dd",
    log: true, // turn off for production
},
livePortrait: livenessTransactionId
}).then((response) => { console.log(`reprocessed: `, response);
return response;}).catch((error) => { console.error(error); });

```

NOTE: Current version of the Regula documentation (<https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/integration/integration-with-face-sdk/#integration-with-face-sdk>) only describes livePortrait as a base64 string, but it's highly recommended to send the liveness transaction id there instead of the image itself to properly follow the zero-trust to mobile concept.

See also: <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/integration/server-side-verification/#step-5-reprocessing-on-web-service>


Analyze results

When all processing is done you need to analyze responses and make a final decision authorize the user or not. It can be simple mapping from the response or can involve complex logic (for instance for some document types errors in fields could be ignored, or you can validate the number of scanned pages/sides depending on the document type and implement retry)

DocReader results

Results are described here: <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/getting-started/results/>

We recommend trying one of the online examples here <https://api.regulaforensics.com/> and explore the Response tab



The screenshot shows the 'Response' tab of the Regula API interface. The 'Status' tab is selected, displaying a JSON object. The JSON structure includes a 'Status' container with 'detailsOptical' and 'detailsRFID' sub-objects, and a 'stopList' array. The 'overallStatus' is 1, indicating success. Other fields include 'buf_length', 'light', 'list_idx', 'page_idx', and 'result_type'.

```

{
  "Status": {
    "detailsOptical": {
      "optical": 1
    },
    "detailsRFID": {
      "rfid": 2
    },
    "stopList": 2
  },
  "buf_length": 84,
  "light": 0,
  "list_idx": 0,
  "page_idx": 0,
  "result_type": 33
}

```

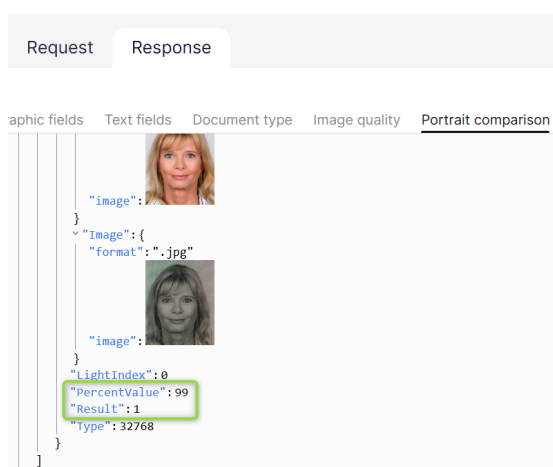
The basic approach – is to check the Status container, overallStatus: 1 indicates success. The main result containers are:

- Status – processing status information

- Images (Graphic fields tab) – contains (cropped) images extracted from the document (pages, portraits, signature, etc.)
- Text – shows all the extracted texts with all kind of details (source, probability, etc.)
- OneCandidate (Document type tab) – holds information about the recognized document type
- ImageQualityCheckList (Image quality tab) – everything about source images quality
- AuthenticityCheckList (Security checks tab) – details about security checks done during the verification process (face comparison results – see the following chapter – are also added to this container, but to another tab)

Face Comparison Results

To see an example of **face comparison results**, try a [Driver License sample](#), or upload your own **document + photo** and check the AuthenticityCheckList container data shown on the Portrait comparison tab.



Freeing resources

Don't forget to free up resources after usage

- DocReader <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/getting-started/deinitialization/>
- FaceSDK <https://docs.regulaforensics.com/develop/face-sdk/mobile/getting-started/initialization/#deinitialization>

```
DocumentReader.deinitializeReader(_ => {}, _ => {});
FaceSDK.deinitialize(_ => {}, _ => {});
```

More technical information

Supplier documentation

For more technical details please check supplier documentation

- DocReader <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/>
- FaceSDK <https://docs.regulaforensics.com/develop/face-sdk/mobile/>

Online demo services

- DocReader <https://api.regulaforensics.com/>
- Face API <https://faceapi.regulaforensics.com/>

Calling API directly

Client libraries for different tech stacks

- DocReader <https://docs.regulaforensics.com/develop/doc-reader-sdk/web-service/development/clients/>
- Face API <https://docs.regulaforensics.com/develop/face-sdk/web-service/development/clients/>

OpenAPI definitions

- DocReader <https://dev.regulaforensics.com/DocumentReader-web-openapi/>
- Face API <https://dev.regulaforensics.com/FaceSDK-web-openapi/>

UI Customization

- DocReader <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/customization/#customize-user-interface>
- FaceSDK <https://docs.regulaforensics.com/develop/face-sdk/mobile/feature-usage/liveness/#ui-customization>

Logging

- DocReader <https://docs.regulaforensics.com/develop/doc-reader-sdk/mobile/configuration/logging/>

Error handling

- FaceSDK <https://docs.regulaforensics.com/develop/face-sdk/mobile/feature-usage/liveness/#error-handling>

Samples

- DocReader
 - Android <https://github.com/regulaforensics/DocumentReader-Android>
 - iOS <https://github.com/regulaforensics/DocumentReader-iOS>
 - React Native <https://github.com/regulaforensics/react-native-document-reader>
 - Flutter <https://github.com/regulaforensics/DocumentReader-Flutter/tree/master/example>
 - Cordova <https://github.com/regulaforensics/DocumentReader-Cordova-Plugin/tree/master/example>
 - Ionic <https://github.com/regulaforensics/DocumentReader-Ionic-Plugin/tree/master/example>
 - .NET MAUI <https://github.com/regulaforensics/Xamarin-DocumentReader>
- FaceSDK
 - <https://docs.regulaforensics.com/develop/face-sdk/mobile/getting-started/samples/>

Change History

Date of Change	Version	Changed By	Summary of Change
February 2025	1.0	Martin Heikkilä, Volodymyr Levchuk	First version

CONFIDENTIAL