Retrieving the User API Token via OAuth

The API Token Retrieval v 20.42

If you are a system integrator and want to integrate eSignAnyWhere, you quickly come to the point where you are dealing with API authentication. While with older eSignAnyWhere API versions it was possible to use alternatives, the REST API v6 requires an integrator who use the user specific api tokens. But as soon as you are using different user accounts in eSAW, it would not be very convinient to ask each user to create his own api Token in the Web UI and configure it in the integration system. Typical cases of such situations are when you have users in the WebUI.

The solution therefore is to retrieve the API tokens for each user automatically. Technically, we are therefore using the OAuth 2.0 protocol, precisely the OAuth Code Grant flow. In this case, eSignAnyWhere is acting as OAuth 2.0 Identity Provider and is providing the user specific API token as bearer token.

This story covers the following:

- Prerequisites
- Application Flow - The Ideal Hook Point
  Use of a hook point when sending an envelope, to retrieve necessary api Tokens
- Basic Implementation
- Additional Considerations

Prerequisites

To allow the authentication to retrieve access credentials from eSignAnyWhere, which acts as OAuth 2.0 Identity Provider, it is required to define on eSignAnyWhere side a configuration allowing to retrieve access credentials. In the terminology of OAuth, this configuration is called an "OAuth Application". eSignAnyWhere allows to define such OAuth Applications on an instance level via the AdminWeb. In case of a SaaS instance, get in touch with Namirial to create the configuration.

To configure the OAuth Application, following data is necessary:

- A name of the integration (This name will be visible for each user (sender) when granting access for the integration)
- Optional: an image, which represents the integration (Will also be shown to the user)
- Optional: a description of the integration
- The redirect URL to which the one-time code during OAuth Code Grant flow is sent (This endpoint has to be implemented as part of the integration tasks; see "Implementation Basics" Step (C) below)
  The full URL (and not just the domain name) is required. The URL must use the HTTPS protocol, and must be reachable from the eSignAnyWhere server(s).

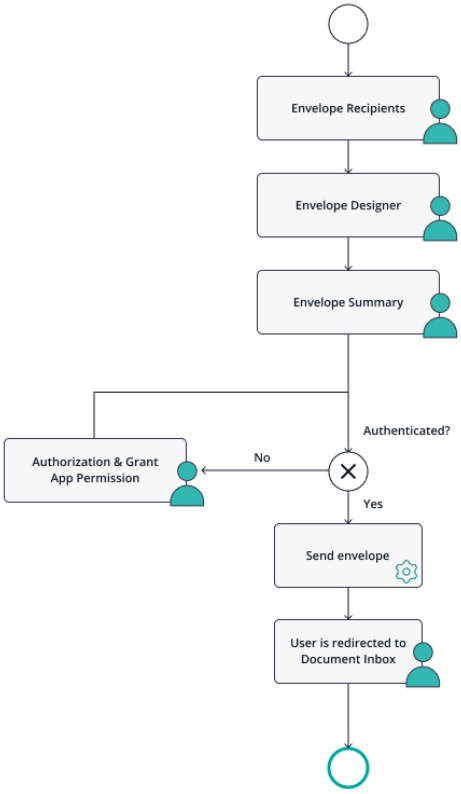
Application Flow - The Ideal Hook Point

We want to ask to grant access in a scenario where the sender user is already involved. Therefore, we identified that the time of sending an envelope usually fits best. With the option to add a "before-send redirect page" in the envelope creator, we are offering a hook point for such activities directly in the envelope creator. When such an URL is configured, the envelope creator changes to a 4-pages mode instead of its typical 3-pages mode: After the recipients page and the designer page, the sender will see the summary page of eSignAnyWhere. In the summary page, instead of the “SEND” button, the sender will see a “NEXT” button because a “before-send redirect url” was configured.

Following placeholders are available for the “before-send redirect url” :

- ##EnvelopeId##
- ##SenderUserId##
- ##OrganizationId##

When the sender clicks the “NEXT” button on the summary page, he/she is redirected to the configured before-send redirect page. We don't really want to show a specific page typically, but only in case the user did not yet grant access to the OAuth Application he will be asked to confirm this access. In standard cases, the envelope create wizard will just send the envelope and close. It's a matter of user experience if you want to show some "final page" with a "send" button or if you accept that the "next" button did implicitly trigger sending the document.

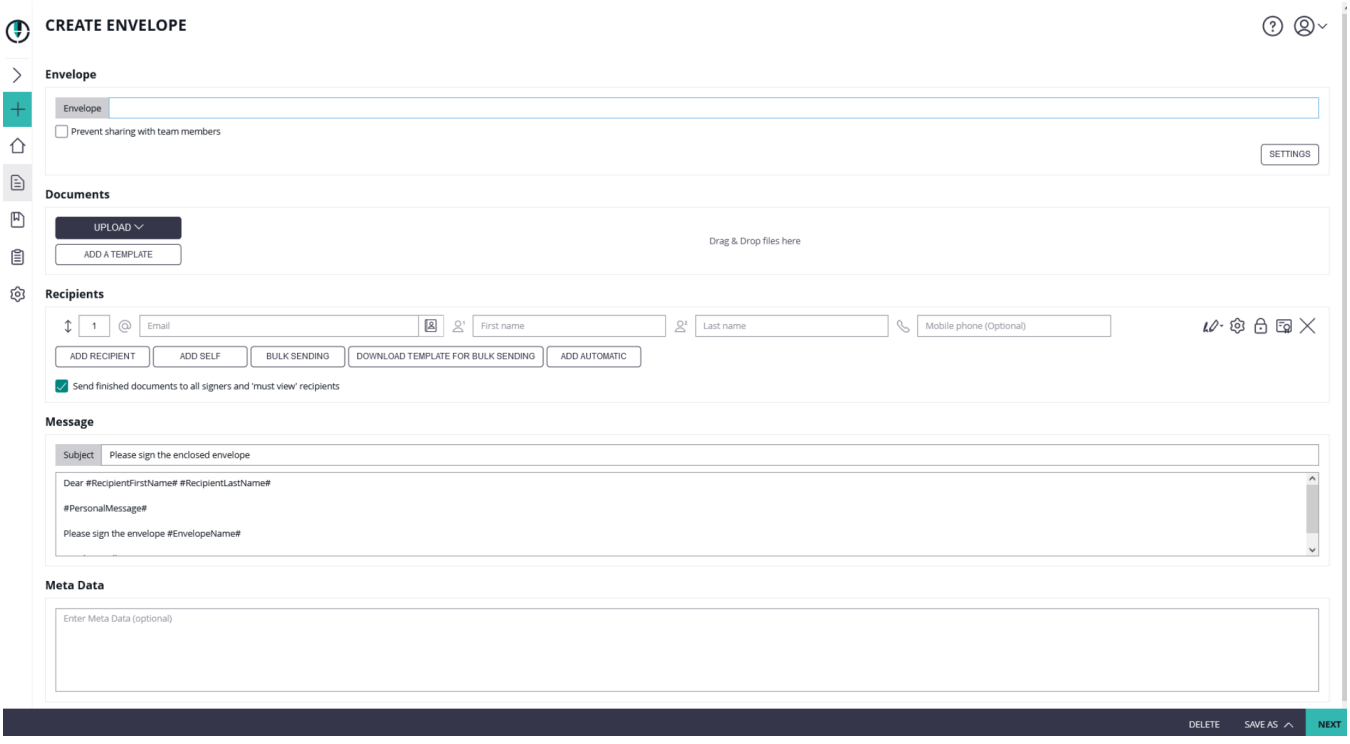


User Experience

This flow will be represented to the user in the following screens:

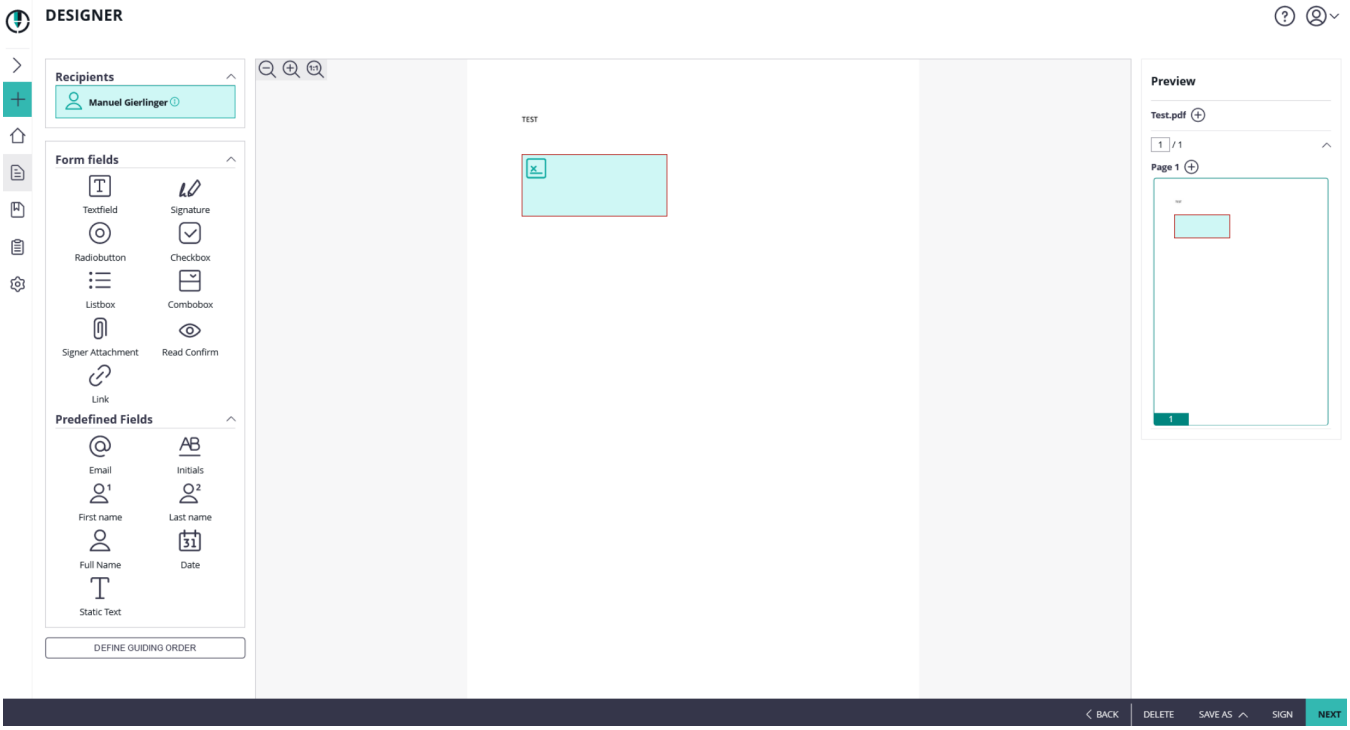
Recipients Page

In the recipients page, the sender selects documents to be signed, and defines the recipients and workflow. This is standard eSignAnyWhere functionality.



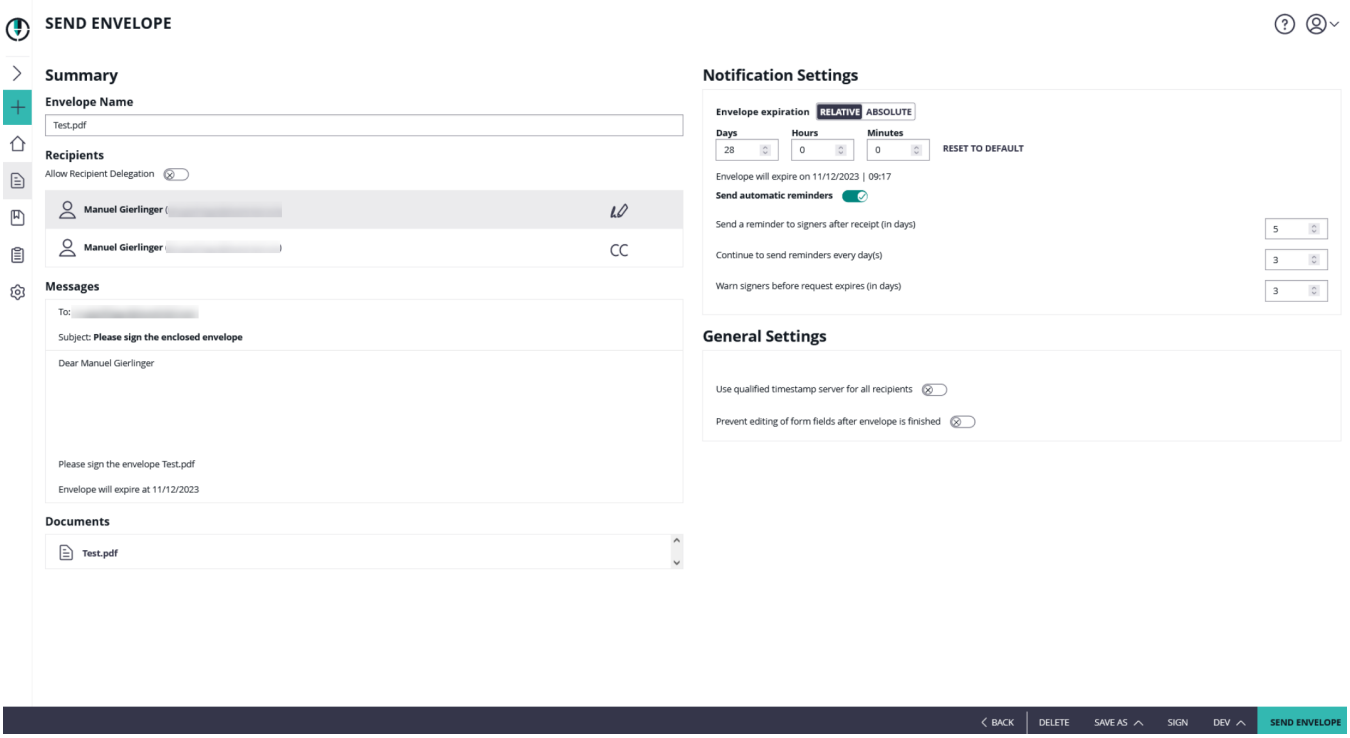
Designer Page

In the designer page, the sender may define or add form fields, signature fields and predefined annotation fields. This is standard eSignAnyWhere functionality.



Summary Page

In the summary page, the sender may adjust some envelope settings such as notifications/reminders.



Authorization Request

Then the user is redirected to the configured redirect page. Whatever custom page should be used, it must return with an HTTP Redirect to that OAuth2 authorization page of eSignAnyWhere when authorization is unknown or invalid (URL is typically https://<esaw-instance>/Auth/Authorize - read below for necessary parameters). E.g. in case of authenticating for a tagging page, the tagging page implementation ensures that the authentication is done first (redirect to authentication, and return to tagging after authentication was passed). If the "beforeSendRedirectUrl" is used solely for authentication, but no specific data input is necessary, an "empty dispatcher page" may be used instead.

If the user is not yet logged in, the Authorize page would start with the standard eSignAnyWhere login form.
As in the context of sending a draft the sender is already authenticated, no login form is shown.

On first usage, the app asks for permissions to grant API access. The user therefore has to give consent in following approval form provided by eSignAnyWhere as part of the OAuth2 Authentication steps:

The name shown in the consent approval form (int the example: "XXXXX DMS tagging service") is part of the configuration of the eSignAnyWhere OAuth Application, so configured centrally per instance in the AdminWeb.

After the consent form was completed (or immediately, in case consent was already approved before), the Authorize page redirects via HTTP 302 to the redirect_uri provided in the GET parameters.
Note that the redirect_uri must be added to the allowlist in the AdminWeb configuration therefore.

In case of using the beforeSendRedirectUri for tagging, the 302 target can be the tagging page. See Document Tagging Scenario - Example showing how to collect metadata for DMS archiving for details.
If using the beforeSendRedirectUri solely for the purpose to collect a bearer token for later usage in a callback handler, the mentioned "empty dispatcher page" could do the following:

- implement the token retrieval in the backend, and store reference envelopeIdsenderUser and senderUserbearerToken
- use draft/send to send the envelope finally
- instead of displaying a tagging page, directly proceed with HTTP 302 to the document inbox

"Done" Status Page (optional)

An optional page that informs that the envelope was sent; or alternatively may contain a button to finally send the envelope. This page could automatically close (i.e. redirect to the document inbox) after some seconds.

Document Inbox

After the envelope is sent, the custom web app should redirect to the eSignAnyWhere document inbox. This is standard eSignAnyWhere functionality; the "done" status page just needs to redirect (e.g. with a HTTP 302) to that page after some seconds or after a button click.

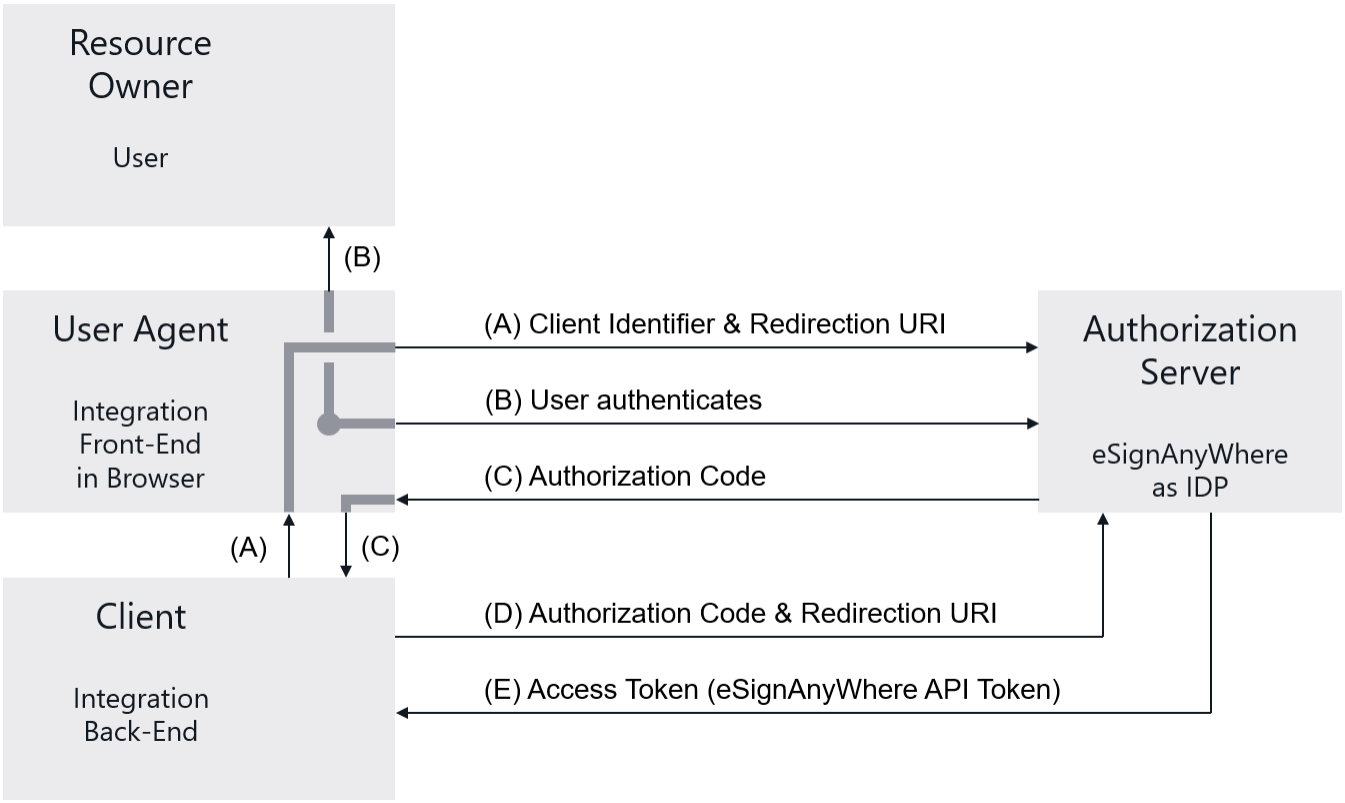
> Note if you want to cancel the settings:
>
> Disable the beforeSendRedirectUrl in Settings-Organization. After all pending envelopes (drafts) have been sent, disable the OAuth Application (prevent log-in via this OAuth App). When disabling the OAuth App, existing drafts still contain the beforeSendRedirectUrl, so the OAuth Application should not be blocked before those drafts have been sent.

Prerequisites

AdminWeb: Create OAuth Application

Implementation Basics

Before implementing the OAuth 2.0 Code Grant, we recommend to read RFC 6749, Chapter 4.1. The diagram in the RFC explains the communication flow. Derived from the diagram in the RFC 6749, the following diagram puts the components in the context of eSignAnyWhere and your integration code:



In Step (A), your front-end therefore redirects to the authorization page provided by the IDP, and provides the client_id and the redirection URI as parameters to the authorization call. The redirection can be implemented via a HTTP 302 redirect or by opening the authorization page in a separate window. Note that OAuth's security concept requires that the front end therefore is not aware of the client_secret. The client_secret should be treated like a password and must never be available in front-end code. The URI of the authorization page is provided to you along with the client_id and client_secret when obtaining the prerequisites. It will typically be something like:

```
https://<esaw-instance>/Auth/Authorize?
response_type=code&client_id=<client_id>&redirect_uri=<redirect_uri>&state=<state>
```

where state is just a random input parameter (see RFC 6749 for details) and the redirect_uri has to be URL-encoded as it is passed on within an URI parameter.

Step (B) does not require any programming. It just visualizes the human interaction where the user is involved and enters his access credentials to the login page and/or grants access permission when using the OAuth application for the first time. As the user is typically already logged in when sending an envelope, he will not be asked to authenticate to eSignAnyWhere again as a valid user session is found.

Step (C) is triggered by the OAuth 2.0 Identity Provider. eSignAnyWhere will therefore just redirect, with a HTTP 302 typically, to the redirection uri provided in Step (A).
Your integration has to provide a front-end page, available at the provided redirect URI, accepting the "authorization code" as URI parameters.
The implementation has to forward that code to the back-end, might wait for receiving a success on step (D) and (E) from the back-end and then redirect to any other front-end page.
Mind to implement a proper error handling, as defined in RFC 6749.

Steps (D) and (E) then have to be imlpemented in the back-end. Using the authorization code retrieved in step (C), the back-end has to call the token endpoint of the OAuth 2.0 Identity Provider. The token endpoint of eSignAnyWhere is provided to you along with the client_id and client_secret when obtaining the prerequisites. In the response, the Access Token will be returned. The access token is the user API token which you can use for further calls to the API as long as the user doesn't revoke access permissions for the OAuth Application.

The token URI endpoint will typically be something like:

```
POST https://<esaw-instance>/ApiToken/Retrieve
```

where the parameters are provided as JSON structure in the POST body:

```
{
  "client_id" : "<client_id>",
  "client_secret" : "<client_secret>",
  "code" : "<code provided in step B>",
  "redirect_uri" : "<redirect_uri_notencoded>"
}
```

eSignAnyWhere does not require to use a nonce parameter, and does not use refresh tokens. Therefore, the token received via above's call sequence will be valid until revoked.

Note that the envelope was not yet sent when a before-send redirect uri is invoked. As there is no "send" button in eSignAnyWhere, use the eSignAnyWhere API methods for the draft with the provided DraftID to send the draft. After sending, we call it an envelope. After sending, which may be done immediately or after a human interaction in the before-send redirect page, return a HTTP 302 or implement some timeout on the clientside website to return to the document inbox by calling its URI directly.

Additional Considerations

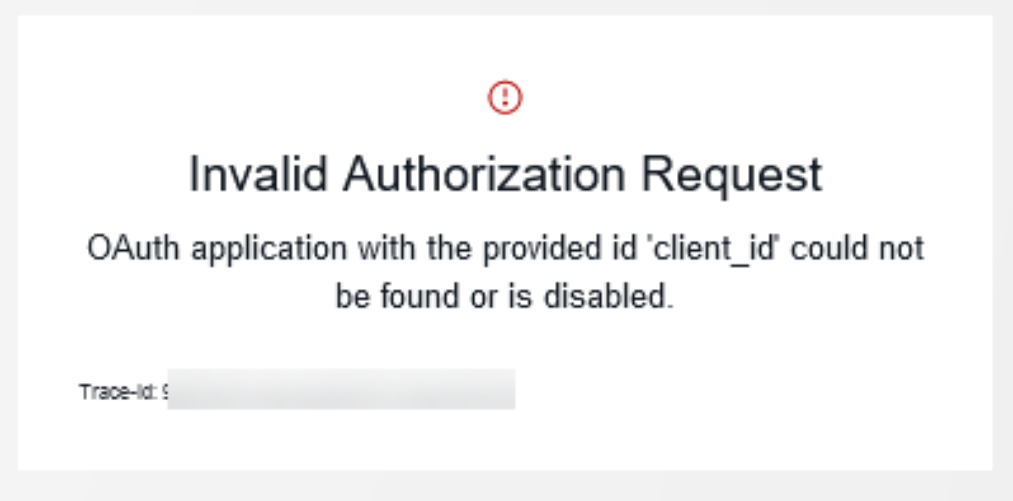
- Mind the differences between "direct usage" and "later usage"
  - Direct usage at the time of sending the envelope allows to use the user API token without persisting it. The time between authentication and using the token is quite short, and it would be a very specific case if the api token was invalidated between authentication and token usage.
  - A typical scenario of "later usage" would be to use the api token e.g. in a callback handler or any other service accessing the envelope at an independent time. In this scenario it is mandatory to store the api token for a later use. The token is per user, so we recommend to store the token with a reference to the sender user and not with an envelope reference. Another token of same user could be reused, instead of requesting separate tokens for each envelope. The different time of processing makes it more likely that the token should be persisted somewhere (see the next point about encryption) or that the token got revoked by the user (another item in the list below).
- Encryption of stored passwords
  A user API token should be treated like a password. In case it has to be persisted for later use, think carefully about proper encryption. See e.g. OWASP Password Storage Cheat Sheet which is pretty clear that in exceptional cases of storing a password - like we have it here - it has to be encrypted.
- What if the user revoked the permission for the OAuth Application
  A user can at any time revoke the access he granted to an OAuth Application. Therefore, it is necessary to think about a proper user experience how to handle "invalidated tokens" - e.g. by suspending processing of this user's background activities until a new user API token was provided; and sending emails which allow the user to re-authenticate via OAuth.

Troubleshooting

Here we are summarizing common FAQ for scenarios where eSignAnyWhere's OAuth 2.0 Identity Provider capabilities are used.

OAuth Authorization Error: ‘redirect_uri’ does not match any of the configured urls

“The provided url ‘redirect_uri’ does not match any of the configured urls of the OAuth application.”:



The exact URL of the web application has to be configured in the OAuth Application configuration, available in the AdminWeb.