Moscow Exchange

# Connection to MOEX WebAPI

Document version 1.3
4.10.2023

# Table of contents

# Connecting to API step-by-step

Steps required to access the Moscow Exchange Web API:

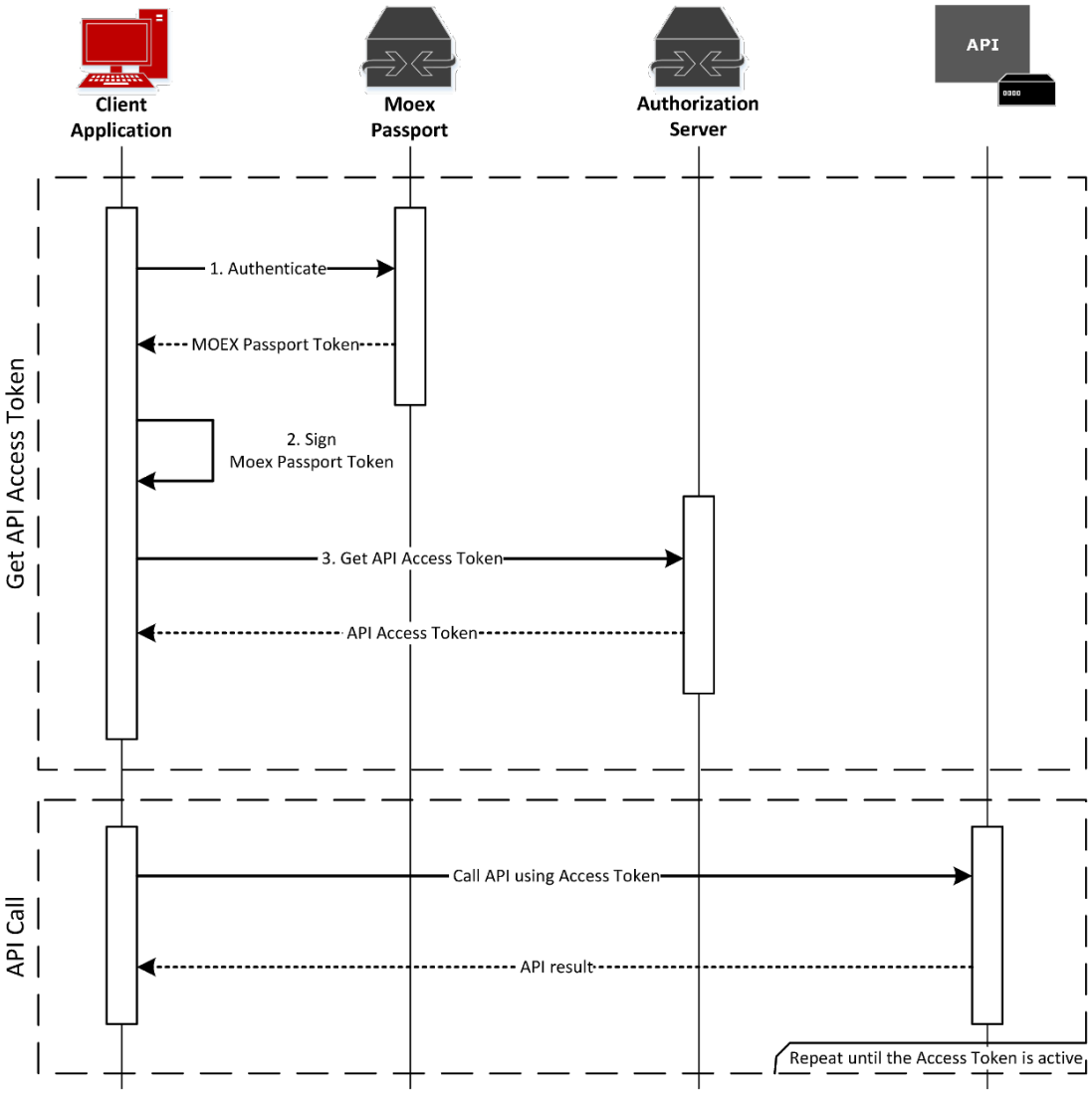
- Create a software application to work with Moscow Exchange APIs. In your software application implement OAuth 2.0 protocol and obtain the access token (for details see below) to call the API functions.
- Test your software at MOEX test environments – contact help@moex.com to get more details and to receive test client_id/client_secret keys and a test electronic signature.
- When ready to go into production contact your personal MOEX client support manager to receive production **client_id** and **client_secret** keys (the unique app ID and security key, the pair will explicitly identify your application when addressing to the API).

> *Please note that a user intended to work with an API should be granted with the appropriate information system usage approval. To obtain the approval, please apply to your client support manager by sending the appropriate application. In addition, the Moscow Exchange Certification Authority is to issue a token certificate registered to the user (owner of the certificate).*

# What is OAuth 2.0?

*OAuth is an open authentication and authorization standard, which provides access to the server resources on behalf the resource owner (another client, or an end user). In addition, OAuth 2.0 provides possibility for end users to set up access to their server resources for third parties, without sharing their credentials.*

# General workflow diagram




# How to obtain the access token

To work with API you are required to obtain the access token by following the steps described below:

1. Obtain your MOEX Passport Token by sending request 'GET' to https://passport.moex.com/en/registration using Basic authentication with the appropriate user credentials. The MOEX Passport Token value will return in a cookie named 'MicexPassportCert'.
2. Using VALIDATA data encryption tool (to generate GOST and RSA signatures), or any other data encryption tool which allows to generate RSA signatures), generate a detached digital signature for the obtained earlier MOEX Passport Token using the appropriate user certificate. For more information on work with data encryption tools please refer to http://moex.com/s1292 (available in Russian only);
3. Send 'POST' request to https://sso.moex.com/auth/realms/SSO/protocol/openid-connect/token, using the following parameter settings (all parameters should be sent using method 'application/x-www-form-urlencoded'):

- **grant_type** – password
- **grant_type_moex** – passport
- **scope** – requested access rights (for online registration, the value is *client_registration*)
- **client_id** – software application ID, provided by your manager
- **client_secret** – security key, provided by your manager
- **certificate** - MOEX Passport Token (obtained earlier at step 1)
- **algorithm** – GOST or RSA value, depending on the signature type used on generating MOEX Passport Token signature.
- **signature** – MOEX Passport Token Base64 digital signature, obtained earlier at step 2.

On successful request, the system returns a JSON object containing the following fields:

- **access_token** – API access token to be used on every API call
- **expires_in** – access token lifetime in seconds
- **refresh_expires_in** – refresh token lifetime in seconds
- **refresh_token** – refresh token. A token using to refresh your access token
- **token_type** – always *Bearer*
- **not-before-policy –** indicates if policy of not using the token before proper time of creation is active ('0' – the policy is inactive)
- **session_state –** identifier of authenticated session
- **scope –** granted access rights

In case of invalid data sent (invalid client_id, or client_secret does not match the client_id, or the digital signature sent does not match the obtained token), the system returns **HTTP Response Code 403**.

## How to use access token

Once you get your access token, you are able to use the token to sign requests you send to the API.

To do that, add the following header to your requests:

*Authorization: Bearer <access_token>*

If your access token is invalid, or expired, the system returns **HTTP Response Code 401**. Once you get the response, you are able to repeat the request for access token following the method described above.

## Testing

To work with API, one should first obtain their access token. To do that, please follow the steps described below:

1. Create a test account at https://passport-test.moex.com/en/registration. Obtain a MOEX Passport Token by sending request 'GET' to https://passport-test.moex.com/authenticate using Basic authentication, with the appropriate user credentials. The MOEX Passport Token value will return in cookie named 'MicexPassportCert'.

2. Write an email with header 'Testing WebAPI <picked service>' to help@moex.com. The email should contain:
    - The email address used for registration at MOEX Passport (see step 1);
    - ID of the firm you use to work at test environment INET_GATEWAY/INETCUR_GATEWAY (if any). If there is no such a firm yet, the appropriate record will be added;
    - Desired certificate type (RSA/GOST) for step 3.

    The reply email will contain:
    - Client test certificate for step 3;
    - **client_id, client_secret** for step 4;
    - Credentials to connect to INET_GATEWAY and INETCUR_GATEWAY, if needed.
3. For GOST digital signatures get the Validata of 6.0 version software from http://moex.com/s1292 (page available in Russian only):
    - Download the appropriate 32 or 64 bit version of Certificate library ZCS. Install it without enabling the TLS component, include the registry storage component if needed. Reboot.
    - Download and install Validata CSP. Start Validata CSP.
    - Start the Certificate library. When started for the first time, chose to recover from a backup and point to the Spr folder of the package received in step 2 above. Or just copy the contents of that folder to C:\Users\<USER>\AppData\Roaming\Validata\zcs\. During the following use select the test certificate. The Certificate library will prompt to insert a certificate media during the next runs. To prepare this media copy the contents of the vdkeys folder to the root of a USB stick or a virtual diskette drive.
    - Use the command line certificate utilities provided with Certificate library and stored at C:\Program Files\Validata\zpki by default. For GOST certificates use the zpki1utl tool.
    - Create a detached digital signature:
      zpki1utl.exe -profile <User> -sign -detached -data <token> -out token.p7d

    For RSA digital signatures get the Validata software from http://moex.com/s1292 (page available in Russian only):
    - Download and install the appropriate 32 or 64 bit Certificate library RCS, version 6.0 or newer.
    - Start the Certificate library. When started for the first time, chose to recover from a backup and point to the Spr folder of the package received in step 2 above. Or just copy the contents of that folder to C:\Users\<USER>\AppData\Roaming\Validata\rcs\. During the following use select the test certificate. On following runs select the UserOrg.rsa key.
        - Download and unpack the command line certificate utilities from http://fs.moex.com/cdp/po/rpkiutlv6.zip. For RSA certificates use the rpki1utl* (32 or 64 bit version) tool.
        - Create a detached digital signature:
          rpki1utl.exe -profile <User> -sign -detached -data <token> -out token.p7d
4. Encode the resulting token in base64 using any tool or a function in your programming language. For example, with a utility shipped with MS Windows:
   certutil -encode token.p7d token.sig
   Note that the final base64 signature must not contain any and of line symbols! So, when using a tool that doesn't have an option to disable EOLs (such as the mentioned certutil) all the EOL symbols must be stripped off.
5. Send request 'POST' to https://sso2.beta.moex.com/auth/realms/SSO/protocol/openid-connect/token, using the following parameter settings (all parameters should be sent using method 'application/x-www-form-urlencoded'):
    - **grant_type** – password
    - **grant_type_moex** – passport

- **scope** – requested access rights (for online registration, the value is *client_registration*)
- **client_id** – software application ID, obtained at step 2
- **client_secret** – security key, obtained at step2
- **certificate** - MOEX Passport Token (obtained at step 1)
- **algorithm** – GOST or RSA value, depending on the signature type used on generating MOEX Passport Token signature, obtained at step 3.
- **signature** – MOEX Passport Token Base64 digital signature, obtained at step 4

General algorithm description of the two-factor authentication:

**MicexPassportCert** = Authenticate_MOEX_Passport (login, password) ;

**Signature** = Validata.create_EDI (**MicexPassportCert** , CLIENT_CERTIFICATE);

**access_token** = HTTP.post (url=https://sso2.beta.moex.com/auth/realms/SSO/protocol/openid-connect/token, parametes= {

grant_type= “password”, grant_type_moex= “passport”, scope “client_registration”,client_id=Client_id, client_secret=Client_secret, certificate= **MicexPassportCert**, algorithm=”GOST”|”RSA”, signature=**Signature**

}

);

On successful request, the system returns a JSON object containing the following fields:

- **access_token** – API access token to be used on every API call
- **expires_in** – access token lifetime in seconds
- **refresh_expires_in** – refresh token lifetime in seconds
- **refresh_token** – refresh token. A token using to refresh your access token
- **token_type** – always *Bearer*
- **not-before-policy –** indicates if policy of not using the token before proper time of creation is active (‘0’ – the policy is inactive)
- **session_state –** identifier of authenticated session
- **scope –** granted access rights

In case of invalid data sent (invalid client_id, or client_secret does not match the client_id, or the digital signature sent does not match the obtained token), the system returns **HTTP Response Code 403**.