The purpose of this document is to provide information about the way the client applications will authenticate a user with the Web API. The goal of the Authentication API is to get a valid authentication token, which will then be passed along every other API request.

The authentication / authorization process involves sending from the client to the web server, sensitive information, such as usernames, passwords and tokens. Therefore, **the client - server communication must be done over HTTPS**.

The base authentication URL is `https://api.collaboard.app/auth`

Flow

The flow of the authentication process is described in the following steps:

- Uses logins to the application using username/password, and the client calls the Web API to get an authorization token.
- If two-factor authentication (2FA) is enabled for the user, the application prompts the user to enter a one-time password (OTP). The client shall exchange the original authorization token along with the OTP code, with a new authorization token that has passed 2FA.
- The application sends the token with every request to the Web API.
- On each request, the Web API validates the token and executes the request, otherwise returns an Unauthorized status.

The authorization flow is also described in the following diagram:

token-authentication-flow.png

Authentication modes

Currently, there are four authentication modes supported by the Web API:

ID	Mode	Description	
1	Username & Password	The user must provide a username and password to login	
2	Username & OTP	The user must provide a username and an OTP code to login	
3	Username & Password with two-factor authentication	The user must provide a username and password to login, and after a successful login, must provide an OTP code to be able to user the Web API methods, as described in the section **Two-Factor Authentication (2FA) with OTP**	

These authentication modes can be used by both local users and external users. In case of an external account and if a password is required, the user shall be authenticated using with the external login provider.

In order to determine the authentication mode a user has selected, the client can call the get authentication mode api endpoint `/api/Authorization/GetAuthenticationMode` and provide the user name required. The server's response will be:

```
{
    "Result": 3,
    "ErrorCode": 0
}
```

, where the result property will contain the authentication mode selected by the user.

The client can use this information to display an appropriate UI to the user, depending on the authentication mode required.

Note: To prevent malicious usage, if the user is not found the api **will return 1** as the result without an error code, so that an attacker will not actually know if the user is registered or not.

Login

The login process consists of authenticating the user with the web server using their username and password, and obtaining a user-specific token, that the client app will then send with every https request to the Web API. If the token is valid, access to the resource will be granted, otherwise the client will get an HTTP Unauthorized status message.

The token format is used is JSON Web Tokens (JWT). More information about their format can be found on the following sites: https://en.wikipedia.org/wiki/JSON_Web_Token https://jwt.io/

The client will call the authentication api endpoint `api/Authorization/Authenticate` using HTTP GET, and will pass the username and password (or OTP code, depending on the user's authentication mode) of the user in the Authorization header using Basic authorization. The server will validate the credentials and, if valid, will return an authorization token `AuthorizationToken` , the number of seconds in which the token will expire `ExpiresIn` , and the user's authentication mode. Also included is a refresh token, that can be used to obtain a new authorization token when this one expires (see section Token Refresh):

```
{
    "AuthorizationToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEZW1vIiw
idGZhIjpudWxsLCJleHAiOjE1NzYwNzg0ODB9._2AzhTVho20t9Gy6op3m_pdZ1Z6Tc48ZYqDVpmzJGQ4",
    "RefreshToken": "eyJ0eXAiOiJKV1QiLciOiJIUzI1NiJ9.eyJ1IjoiZHN0ZWxsZhbHNlLCJleHAi
OjE1ODYwMTQ3ODV9.8ZFtW8jd4O2yNXJm9AaHVj2pwjYjjAU",
    "ExpiresIn": 3600,
    "AuthenticationMode": 1,
    "ErrorCode": 0
}
```

The authorization token contains information about the current user, and shall be passed in the Authorization header of any subsequent API request, in the form of a Bearer token:

`Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEZW1vIiwidGZhIjpmYWxzZSwiZXhwIjoxNTc1MzY5MTQ2fQ._cfJnh7Jl5TMGUw9fyzyrSoJwfLZXUDkcCESQ93M11U`

The server will validate the authorization token and proceed to return the results of the API method call, or HTTP Unauthorized status. Because the Web API is stateless, the authorization token must be provided with every request.

If the login call did not succeed, the server will return HTTP Unauthorized status, and the response will contain the error that occurred. Some of there are:

Error	Meaning
127	The username or password provided was not correct
130	The user is not yet verified

NOTE: If the user has opted for two-factor authentication, the refresh token will be null. The user will need to complete the authentication process in order to get a valid refresh token.

Authenticated user details

In order to get the details of the currently authenticated user, the client call the `api/Authorization/GetAuthenticatedUser` endpoint using HTTP GET and passing the authorization token in the Authorization header using Bearer authentication. The web api will return a JSON object that contains the authenticated user's details:

```
{
    "Result": {
        "UserName": "test@test.com",
        "FirstName": "Test",
        "LastName": "User",
        "Email": "test@test.com",
        "PhoneNumber": "12345",
        "Language": "en-US",
        "PhotoUrl": "http://localhost/api/Authorization/GetUserPhoto?token=eyJ1c2VyIjoiZHN0ZWxsYWtpc0BnbWFpbC5jb20iLCJ0eXBlIjoianBlZyIsImhhc2giOiI2MkRCRTk5RjgzNjdFQjMyNEE4NjUxQzE2RkE5MzJBMCJ9",
        "DateCreated": "2020-04-15T11:49:49.59",
        "DateLastLogin": "2020-08-21T15:31:31.503",
        "TermsOfServiceAccepted": true
    },
    "ErrorCode": 0
}
```

The user's profile image url is returned in field `PhotoUrl`.

The value for the Country property is a 2 letter ISO code of the country (https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes)

`DateCreated` is the date that the user was first registered, `DateLastLogin` is the date that the user last logged in successfully.

Token refresh

When the client authenticates with the Web API, the number of seconds until the token expires is returned in the property `ExpiresIn` of the JSON response.

If the client needs to extend the authorization time window, it should issue an HTTP GET request in the refresh authorization token endpoint `api/Authorization/RefreshToken`, providing in the Authorization header the already obtained refresh token from the Login step:

The authorization token contains information about the current user, and shall be passed in the Authorization header of any subsequent API request, in the form of a Bearer token:

```
Bearer
eyJ0eXAiOiJKV1QiLciOiJIUzI1NiJ9.eyJ1IjoiZHN0ZWxsZhbHNlLCJleHAiOjE1ODYwMTQ3ODV9.8ZFtW8jd
4O2yNXJm9AaHVj2pwjYjjAU
```

If the refresh token is valid, the response will contain the new authorization and refresh tokens, along with the extra details for the expiration time and 2FA, similar to the login process.

NOTE: The refresh token must be kept a secret and stored in a secure location, because it is in fact a way of obtaining an access token for a user without the need to login. It is the client app's responsibility to ensure that these safety standards are met. If there is no way of securely storing the refresh token, it would be best if the token is only kept in-memory (although this would require the user to login each time they open the client app).

Two-Factor Authentication (2FA) with OTP

In case the user opts for two-factor authentication (2FA), the client app should prompt him to enter the one-time code (OTP) that he has obtained using a client authenticator app (such as Google Authenticator or Microsoft Authenticator), or that was sent to him via email from the web api.

After the Login process, and if the server responds that the user has enabled 2FA, the client should prompt the user to enter an OTP code, and call the 2FA verification api endpoint `api/Authorization/ValidateUser2FA` using HTTP POST, passing the authorization token received by the login process, and a JSON object with the OTP Code:

```
{
    "OTPCode": "193825"
}
```

The server will validate the OTP code provided, and it will return new authorization & refresh tokens that are now 2FA-verified. The client shall use the new authorization token for every subsequent API request. If the OTP code cannot be verified, the server will return HTTP Unauthorized status.

```
{
    "AuthorizationToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEZW1vIiw
idGZhIjp0cnVlLCJleHAiOjE1NzUzNzkxMzR9.eT6XjaZXKBILdIXb5f_xh5E3Rs1jxR66M3l0KOPGBlo",
    "RefreshToken": "eyJ0eXAiOiJKV1QiLciOiJIUzI1NiJ9.eyJ1IjoiZHN0ZWxsZhbHNlLCJleHAi
OjE1ODYwMTQ3ODV9.8ZFtW8jd4O2yNXJm9AaHVj2pwjYjjAU",
    "ExpiresIn": 3600,
    "AuthenticationMode": 3,
    "ErrorCode": 0
}
```

Getting OTP codes without the need of an authenticator app

In case the user is using OTPs in his authentication mode, and he does not have access to an authenticator app, the system can generate OTP codes for him. To do so, the client shall call the send otp token api endpoint `/api/Authorization/SendUserOTPToken`, and provide the username of the user and the messaging platform that will be used to send the message (Email or SMS):

```
{
        "User": "test@test.com",
        "MessagingPlatform": "Email"
}
```

If the messaging platform is omitted, the OTP code will be sent to the user via email.

If the user is registered & validated in the system and he has enabled OTP codes, the web api will send to his registered email address an OTP code.