# NoteMark API Milestone #1

This will serve as a documentation for all the API endpoints needed to build NoteMark milestone #1. Please note that this API has been written with care, however if you still find some kind of bug, it would be appreciated if you could report it in Discord.

**The base URL for the NoteMark API is** https://notemark.pl-coding.com

---

# Rate Limiting

To prevent misuse, the API has a rate limit of **400 requests/hour.** Once that limit is reached, the API will respond with 429 Too Many Requests.

---

# Authentication

## Mobile Dev Campus Email

All endpoints require your email used for the Mobile Dev Campus subscription as a form of authentication. It needs to be attached as `X-User-Email` header, like this:

| Header name | Header value |
| --- | --- |
| X-User-Email | <your email> |

## JWT Refresh Token Authentication

All authenticated routes work with refresh token authentication. Here's an exact breakdown of how this works.

### What is JWT Authentication?

JWT (JSON Web Token) is a secure way to verify who you are when using an API. Think of it like a digital ID card that proves the identity of a user to the server.

## The Two-Token System

NoteMark's authentication system uses **two types of tokens**:

**1. Access Token**

- **Purpose**: Your "daily pass" to access protected API endpoints
- **Lifespan**: Short (15 minutes)
- **Usage**: Sent as a header with every API request that requires authentication
- **Security**: If stolen, it expires quickly, limiting damage

**2. Refresh Token**

- **Purpose**: Your "master key" to get new access tokens
- **Lifespan**: Long (30 days)
- **Usage**: Only used to request new access tokens
- **Security**: Stored securely and can be revoked if compromised

## How the Authentication Flow Works

1. As a first step, users have to register a new account. This requires a username, email and password. This step is not yet related to the use of tokens.
2. Once an account has been created, you can log in with your pair of email and password. For a successful login, the login endpoint will respond with a **pair of access and refresh token.**
3. Both access and refresh token now need to be saved locally on the client device.
4. For every request made to an authenticated API endpoint, attach the access token from local preferences as an Authorization header in the following format (replace `<your access token>` with your token):

| Header name | Header value |
| --- | --- |
| Authorization | Bearer <your access token> |

5. If you attempt to make a request with an expired access token (happens after 15min), the API will respond with 401 Unauthorized. Whenever you get this error code, it's a sign for you to get a new short-lived access token by

using the locally saved refresh token. Simply make a request to the /api/auth/refresh endpoint while attaching the refresh token to get a new access token you can then use to retry the failed request with. **This refresh endpoint will give you a new pair of access and refresh token which you need to both override locally. The old refresh token will be invalidated.**

6. If at some point the refresh token expires (happens after not refreshing with it for > 30 days), the session is considered expired. This can be detected by looking at the response code from the /api/auth/refresh endpoint - if it responds with 401, the refresh token is invalid. In that case, the user would need to be forced logged out, the local tokens must be cleared and a new pair of tokens must be acquired by letting them log in again with their credentials.

👉 **Testing the OAuth mechanism**

In order to test your token refresh mechanism without always having to wait 15min for the access tokens to expire, you can simply attach this header to your requests which makes all issued access tokens valid for only 30s - perfect for debugging!

| Header name | Header value |
| --- | --- |
| Debug | true |

💡 **Good to Know**

We know that this logic can seem overwhelming if you haven't worked with this type of authentication yet. That's why at the bottom of these docs, you can find helpful code snippets and resources to assist with implementing this mechanism. And you're of course welcome to ask your questions on Discord!

# Routes

**Registration**

Endpoint: `/api/auth/register`

Method: POST

Description: Registers a new user account.

Request body:

```
{
  "username": "<username>" (String),
  "email": "<email>" (String),
  "password": "<password>" (String)
}
```

**Login**

Endpoint: `/api/auth/login`

Method: POST

Description: Log in with an existing account.

Request body:

```
{
  "email": "<email>" (String),
  "password": "<password>" (String)
}
```

Successful login response:

```
{
  "accessToken": "<access token>" (String),
  "refreshToken": "<refresh token>" (String)
}
```

Invalid credentials response: Status code 401 without a body.

### Token Refresh

Endpoint: `/api/auth/refresh`

Method: POST

Description: Get a new access token by providing your refresh token.

Request body:

```
{
  "refreshToken": "<your refresh token>" (String)
}
```

Success response:

```
{
  "accessToken": "<new access token>" (String),
  "refreshToken": "<new refresh token>" (String)
}
```

## Error Codes

The HTTP status codes can tell you a lot about what went wrong with a request. The NoteMark API sticks to common conventions for these status codes - here some examples and what they mean:

### HTTP 400 Bad Request

Returned when something about your request is not in the right format. You may be missing a mandatory request parameter or the JSON structure of your request body is not in the format the server expects (e.g. due to a typo in the keys). Simply double check your request with these API docs.

### HTTP 401 Unauthorized

This simply says that authentication was not successful. Depending on the context, this can mean the provided email/password combination is incorrect or that your attached access token is not valid (anymore).

### HTTP 405 Method Not Allowed

This means you're using the wrong HTTP method, for example a POST request to an endpoint that expected a GET request.

### HTTP 409 Conflict

Your request itself was valid, but it led to a server-side conflict. This could be returned for example if you try to register with an email that already exists or if you try to insert an item with an ID that already exists.

### HTTP 429 Too Many Requests

You've hit the rate limit of 400 requests per hour. Simply wait a bit and try again 🙂

# Further Resources

### Automatic token refresh with Ktor

It's recommended to have a mechanism in place that automatically refreshes the access token once a request responds with 401. Using common networking clients like Ktor makes this fairly easy.

1. Add Ktor's auth plugin to your `libs.versions.toml` as well as your `build.gradle.kts`:

```
[versions]
ktor = "3.0.0" (or latest)

[libraries]
ktor-client-auth = { module = "io.ktor:ktor-client-auth", version.ref = "ktor" }
# other Ktor dependencies
```

2. Install the plugin into your HTTP client and add the refresh logic:

```
HttpClient(CIO) {
    // ...
    install(Auth) {
        bearer {
            loadTokens {
                val tokenPair = sessionStorage.get()
                BearerTokens(
                    accessToken = tokenPair?.accessToken ?: "",
                    refreshToken = tokenPair?.refreshToken ?: ""
                )
            }
            refreshTokens {
                    // 1. Retrieve current pair of tokens
                val tokenPair = sessionStorage.get()

                // 2. Use refresh token to get new access token
                val response = client.post(
                    urlString = "https://notemark.pl-coding.com/api/auth/refresh",
                ) {
                        contentType(ContentType.Application.Json)
                        setBody(RefreshTokenRequest(
                        refreshToken = tokenPair?.refreshToken ?: ""
                    ))
                    markAsRefreshTokenRequest()

                    // Add this, if you want to test your refresh mechanism.
                    // This makes tokens valid for only 30s.
                    header("Debug", true)
                }

                if(response is Result.Success) {
                    // 3. Update session storage with new pair of tokens
                    // (requires adjustment to your local storage)
                                sessionStorage.update(response)

                    BearerTokens(
                        accessToken = newAuthInfo.accessToken,
                        refreshToken = newAuthInfo.refreshToken
```

```
                )
            } else {
                BearerTokens(
                    accessToken = "",
                    refreshToken = ""
                )
            }
        }
    }
}
```

3. That's it! Any request that returns 401 now will trigger this flow implemented in `refreshTokens`.

**JWT Token Playground**

URL: https://jwt.io/

Description: This website can help you to understand JWTs a bit better and how they store information about the logged in user.