Token refresh API — MindEase

Audience : Frontend developer (Tejas)
Base URL : `http://localhost:8000` (use production base in prod)

Endpoint : `POST /api/auth/token/refresh/`

Summary

Use this endpoint to obtain a **fresh access token** when the short-lived access token has expired. The server:

- **reads the refresh token** from the refresh cookie (preferred) *or* from the JSON body

  `{ "refresh": "<token>" }`.

- **validates** the refresh token.

- **on success** sets a new **access_token** cookie (short-lived). If your `SIMPLE_JWT` is configured with `"ROTATE_REFRESH_TOKENS": True`, the server also issues and sets a **rotated refresh_token** cookie and may blacklist the old refresh.

- **Returns :** `200 OK` with a small JSON message (no tokens in JSON — tokens are stored in HttpOnly cookies).

Request headers

Content-Type: application/json
Accept: application/json

Important for browsers : include `credentials: "include"` so cookies are sent and accepted.

Request body (optional)

You *do not* need to send a body if the refresh token is present in cookies. If you prefer to send it explicitly (for example, mobile clients or if cookies are not available), POST:

```
{
  "refresh": "<refresh_token>"
}
```

If the cookie value was stored like `"Bearer <token>"`, the server will normalize it.

Success response

Status: `200 OK`
Body:

```
{
  "detail": "Token refreshed."
}
```

Side effects (HTTP headers) :

- `Set-Cookie : access_token=<NEW_ACCESS_JWT>; HttpOnly; Path=/; Max-Age=<seconds>; SameSite=<...>; Secure=<...>`
- If rotation enabled : `Set-Cookie: refresh_token=<NEW_REFRESH_JWT>; HttpOnly; Path=/; Max-Age=<seconds>; SameSite=<...>; Secure=<...>`
- If rotation is not enabled: the same refresh cookie may be re-set (same token string) or left unchanged depending on implementation.

No token values are returned in JSON (cookies are `HttpOnly`).

Error cases & status codes

- **401 Unauthorized** — missing refresh token:

{"detail": "Refresh token not provided."}

- **401 Unauthorized** — invalid/expired refresh token:

{"detail": "Invalid or expired refresh token."}

- **401 Unauthorized** — token user invalid:

{"detail": "Invalid token user."}

- **500 / other** — unexpected server error (rare). Frontend should be treated as transient and surface friendly error.

Typical frontend flow (recommended)

Whenever User try to access the protected routes then this frontend developer should follow this flow.

1. Frontend makes protected request with `credentials: "include"`.
2. If server responds `200` — good.
3. If server responds `401` with message about expired access token (or `"Invalid or expired access token."`), **do not** immediately send user to login. Instead:
    - Call `POST /api/auth/token/refresh/` with `credentials: "include"`.
    - If refresh call returns `200`:
        - The server sets a new `access_token` cookie (and possibly rotated refresh cookie).

        - Retry the original protected request (again with `credentials: "include"`).
    - If refresh call returns `401` (refresh invalid/expired):
        - Redirect user to login (session ended).
4. Always avoid multiple concurrent refresh calls from multiple tabs — either:
    - serialize refresh attempts client-side (single-flight), or
    - let server rotate and handle idempotency carefully. Prefer client single-flight.

Browser fetch examples

(A). Call refresh using cookie (recommended)

```
// call this when you detect access token expired
fetch("http://localhost:8000/api/auth/token/refresh/", {
  method: "POST",
  credentials: "include",   // REQUIRED so server reads cookie and sets new cookies
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({})  // body optional if cookie present
})
.then(async res => {
  const json = await res.json().catch(()=>({}));
  console.log("refresh status:", res.status, json);
  if (res.status === 200) {
    // retry original request (with credentials: 'include')
  } else {
    // refresh failed -> send user to login page
  }
})
.catch(err => console.error("Network error:", err));
```

(B). Call refresh by sending token in body (cookie-less clients)

```
fetch("http://localhost:8000/api/auth/token/refresh/", {
  method: "POST",
  credentials: "include",   // optional here; cookie not required if sending token in body
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ "refresh": "<REFRESH_TOKEN_STRING>" })
})
.then(r => r.json().then(b => console.log(r.status, b)));
```