# How to call Public APIs vs Protected APIs (short)

- **Public API** : any client can call it. Example : `GET /api/resources/`
    - Use `fetch(url)` (no special cookie handling required). You may include `credentials: "include"` if you want cookies sent.
- **Protected API** : requires a valid **access token**. In this system the server stores tokens in **HttpOnly cookies**:
    - `access_token` (short lived) and `refresh_token` (longer lived).
    - Frontend must always call protected endpoints with `credentials: "include"`.
    - If a protected request returns `401` because the access token expired, call the refresh endpoint `POST /api/auth/token/refresh/` (also with `credentials: "include"`). On success server sets a new `access_token` cookie — then retry original request.
    - If refresh fails (401), redirect user to login.

---

# Minimal frontend helper (paste into browser console or include in app)

This helper :

- Adds `credentials: "include"` automatically.
- On `401` it tries to refresh once (single-flight across concurrent calls).

- Retries the original request after successful refresh.
- If refresh fails, it returns the failed response so caller can redirect to login.

```
// Simple single-file helper. Paste into DevTools console or include in frontend.
(() => {
  // change if your backend base differs
  const API_BASE = "http://localhost:8000";
  const REFRESH_URL = API_BASE + "/api/auth/token/refresh/";

  // single-flight refresh promise shared across requests
  let refreshPromise = null;

  async function doRefresh() {
    if (refreshPromise) return refreshPromise; // reuse
    refreshPromise = (async () => {
      const res = await fetch(REFRESH_URL, {
        method: "POST",
        credentials: "include",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({}) // token read from cookie server-side
      }).catch(e => ({ ok: false, status: 0, error: e }));

      // Clear promise after finished so next refresh may start later
      refreshPromise = null;
      return res;
    })();
    return refreshPromise;
  }

  /**
   * fetchWithAuth(url, options)
   * - automatically uses credentials: 'include'
   * - if response.status === 401 it attempts token refresh once and retries
   * - returns final Response object (not JSON); caller should inspect status and call res.json()
   */
  async function fetchWithAuth(url, options = {}) {
    options = Object.assign({}, options);
    options.credentials = "include";
    options.headers = Object.assign({}, options.headers || {}, { "Accept": "application/json" });

    // first attempt
    let res = await fetch(url, options);
```

```
    if (res.status !== 401) return res;

    // got 401 -> try refresh
    const refreshRes = await doRefresh();

    // If refresh request failed (network) or returned non-200 -> return original 401
    if (!refreshRes || refreshRes.status !== 200) {
      return res; // caller sees 401 and can redirect to login
    }

    // refresh succeeded -> retry original request once
    // ensure we don't enter infinite loop: only one retry
    res = await fetch(url, options);
    return res;
  }

  // Expose helpers globally for console testing
  window.fetchWithAuth = fetchWithAuth;
  window._authHelpers = { doRefresh, REFRESH_URL };

  console.log("fetchWithAuth helper installed. Use fetchWithAuth(url, opts).");
})();
```

## Examples to run in the browser console

- Login (so server sets cookies):

```
fetch("http://localhost:8000/api/auth/login/", {
  method: "POST",
  credentials: "include",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ email: "paras0mani6@example.com", password: "CounsellorPass1!" })
}).then(r => r.json().then(b => console.log(r.status, b)));
```

- Call protected GET:

```
fetchWithAuth("http://localhost:8000/api/auth/profile/", { method: "GET" })
  .then(async res => {
    console.log("status:", res.status);
    const body = await res.json().catch(()=>({}));
```

```
  console.log(body);
  if (res.status === 401) {
    // not refreshed -> send to login
    window.location = "/login";
  }
});
```

- Call protected PATCH (update profile):

```
fetchWithAuth("http://localhost:8000/api/auth/profile/", {
  method: "PATCH",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ first_name: "Paras", bio: "I help people." })
})
.then(async res => console.log(res.status, await res.json().catch(()=>({}))));
```

- Call public API (resources):

```
fetch("http://localhost:8000/api/resources/?type=video")
  .then(r => r.json()).then(console.log);
```

---

# Recommended client-side refresh flow (short)

When User Wants To Access The Protected Routes Then Frontend Developer Has to Follow This :

1. Try the protected request with `credentials: "include"`.
2. If server returns `200` → done.
3. If server returns `401` → call `POST /api/auth/token/refresh/` with `credentials: "include"`.

4. If refresh returns `200` → server set new access cookie; retry the original request (step 1).

5. If refresh returns `401` → session ended → redirect to login page.

Use a single-flight refresh (see helper) to avoid multiple tabs/requests issuing many refreshes.

---

# Endpoint list (what's public vs protected)

- Replace `localhost:8000` with your production base when deploying.

**Public (no auth required) :**

- `POST /api/auth/signup/` — create account (created users `is_active=False`).
    - Success: `201 Created` + message.
- `POST /api/auth/verify-email/` — verify email token.
    - Body: `{ "token": "<raw_token>" }`.
- `POST /api/auth/resend-verification/` — request resend; throttled.
    - Body: `{ "email":"..." }`.
- `POST /api/auth/login/` — returns user JSON + sets HttpOnly cookies (`access_token`, `refresh_token`).
    - Use `credentials : "include"` on fetch to store cookies.
- `POST /api/auth/token/refresh/` — refresh access token; reads refresh cookie (or body).
    - On success `200` and server sets a new access cookie (and rotated refresh cookie if enabled).

- `POST /api/auth/logout/` — clears cookies and optionally blacklists refresh; idempotent.

**Protected (require access token / cookie)**

- `GET|PUT|PATCH /api/auth/profile/` — get & update current user profile (Client/Counsellor). **Protected**. Use `fetchWithAuth`.

**Public Content / Search**

- `GET /api/resources/` — list external resources (articles/videos/pdfs). Public.
- (Search counsellors) — if you added a search API (example path):
    - `GET /api/search/counsellors/?q=<name>&specialization=<name>&min_fee=0&max_fee=500` — typically **public** for clients to search.

If you want a definitive list of all endpoints in your project, run `python manage.py show_urls` (or use `django-extensions` `show_urls`) — I can provide exact commands if you want.

---

# Error handling mapping (frontend guide)

- `200` → success (handle response).
- `400` → validation error — show field errors to user (JSON will contain field-specific messages).
- `401` → authentication required / invalid token:
    - For protected calls: try refresh flow (helper does this). If refresh fails, redirect to login.

    - For login/refresh endpoints: show message (invalid credentials / session expired).

- `403` → forbidden (user authenticated but lacks permission) — show "Not allowed".
- `429` → throttled — show message to wait and optionally use `Retry-After` header.
- `500` → server error — show friendly "Something went wrong, try again later".