Logout API — MindEase

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

Endpoint : `POST /api/auth/logout/`

Summary

Idempotent endpoint that signs a user out :

- **Clears** the browser cookies `access_token` and `refresh_token` (HttpOnly cookies).
- **Attempts** to blacklist the provided refresh token (if `rest_framework_simplejwt.token_blacklist` is installed).
- Supports `"all": true` to invalidate **all** sessions for that user (increments `token_version` and tries to blacklist outstanding refresh tokens).
- Always returns `200 OK` on success (safe to call repeatedly). Frontend should remove any client-side cached user state on success.

Important : call with `credentials: "include"` when using cookies.

Request headers

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

Request body (JSON) — optional fields

```
{
  "refresh": "<refresh_token>",   // optional : override cookie
  "all": true                     // optional : logout all sessions for the user
}
```

- If a refresh cookie is present (`REFRESH_COOKIE_NAME`), the server prefers that over `refresh` body.
- `all` may be `true` (boolean), `"true"` (string) or `1` — server will treat these as true.

Behaviour / Flow (detailed)

1. **Get refresh token** : server checks cookie `refresh_token` (default name) first, then `request.data["refresh"]`.
2. **Always clear cookies** : response will delete both `access_token` and `refresh_token` cookies (idempotent).
3. **If a refresh token is supplied and valid** :
    - Try to **blacklist** that refresh token (if token_blacklist app installed). If blacklist is not available, ignore the error.
4. **If `all` is true and refresh valid** :
    - Extract user id from the refresh token.
    - Call `user.increment_token_version()` to invalidate previously issued access tokens (server-side check).
    - Optionally call `user.blacklist_all_refresh_tokens()` if available to blacklist outstanding refresh tokens.
5. **Return** `200 OK` and a small JSON `{ "detail": "Logged out." }` regardless of token validity. This avoids token-enumeration and keeps the UX simple.

Successful response

Status: `200 OK`
Body:

```
{
  "detail": "Logged out."
}
```

Also the response will include `Set-Cookie` headers that clear `access_token` and `refresh_token`.

Example error cases (note : API is intentionally forgiving)

- If the refresh token is missing or invalid → server still clears cookies and returns `200 OK`.
  (This is by design: logout is idempotent and should not leak token state.)
- If blacklisting fails (blacklist app not installed) → server ignores and still returns `200 OK`.

If you prefer strict errors (e.g., `401` on missing token), we can change behavior — but idempotent success is recommended.

Frontend integration tips

- Always call logout with `credentials: "include"` so the server can read cookies and return cookie-clearing headers.
- On `200 OK` clear any client-side state (Redux / local state) and redirect to the login/landing page.

Browser console (fetch) example — normal logout

```
fetch("http://localhost:8000/api/auth/logout/", {
  method: "POST",
  credentials: "include",
```

```
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({})
})
.then(async res => {
  const json = await res.json().catch(()=>({}));
  console.log("status:", res.status, json);
  // Client: clear user state + redirect to login
})
.catch(err => console.error("Network error", err));
```

Logout-all (invalidate everywhere)

```
fetch("http://localhost:8000/api/auth/logout/", {
  method: "POST",
  credentials: "include",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ "all": true })
})
.then(r => r.json().then(b => console.log(r.status, b)));
```

Status codes you will see

- `200 OK` — logout performed (cookies cleared). Always expected on normal usage.
- `4xx/5xx` — network/server error (rare). Example : `500` if the server throws an unexpected exception.