

Login API — MindEase

Audience : Frontend developer (Tejas)

Base URL : `http://localhost:8000` (use production base in prod)

Endpoint : `POST /api/auth/login/`

Summary

Authenticate a user with `email` + `password`. On success the server:

- sets two `HttpOnly` cookies (`access_token`, `refresh_token`) — *no tokens are returned in JSON*
 - returns `200 OK` and a small `user` object in the JSON body (safe to store in client state). Frontend must use `credentials: "include"` to allow cookies to be set.
-

Request headers (required)

Content-Type : `application/json`

Accept : `application/json`

Request body (JSON)

```
{
  "email": "user@example.com",
  "password": "UserPassword123!"
}
```

Field rules / validations (server-side)

- `Email` : required, valid email

- **Password** : required
-

Successful response (200 OK)

- Cookies set (HttpOnly) — **access_token** (short-lived) and **refresh_token** (longer-lived)
- JSON body includes user details (no token value in JSON).

Example response headers (server sets cookies) :

```
Set-Cookie : access_token=<JWT_ACCESS>; Max-Age=120; HttpOnly; Path=/; SameSite=Lax
Set-Cookie : refresh_token=<JWT_REFRESH>; Max-Age=604800; HttpOnly; Path=/;
SameSite=Lax
```

Example JSON body :

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "detail": "Login successful.",
  "user": {
    "id": 42,
    "email": "paras0mani6@example.com",
    "first_name": "Paras",
    "last_name": "Mani",
    "role": "client"
  }
}
```

Important : Cookies are **HttpOnly** and cannot be read from JavaScript. Keep user data you need in application state (redux/context).

Common error cases

401 Unauthorized — invalid credentials

```
HTTP/1.1 401 Unauthorized
{
  "Detail": "Invalid credentials."
}
```

401 Unauthorized — account inactive / email not verified

If user exists but `is_active` is `False` (email not verified), return :

```
HTTP/1.1 401 Unauthorized
{
  "Detail": "Account inactive. Please verify your email before logging in."
}
```

Frontend should show a friendly prompt + option to resend verification.

400 Bad Request — missing fields / validation

```
HTTP/1.1 400 Bad Request
{
  "email": ["This field is required."],
  "password": ["This field is required."]
}
```

429 Too Many Requests — throttled

If login attempts are rate-limited:

```
HTTP/1.1 429 Too Many Requests
{
  "detail": "Request was throttled. Expected available in 60 seconds."
}
```

Check `Retry-After` header for seconds.

500 Internal Server Error

Generic server error:

```
HTTP/1.1 500 Internal Server Error
{ "detail": "Internal server error." }
```

JWT / cookie behaviour (brief)

- **Access_token** : short-lived (configured via `SIMPLE_JWT["ACCESS_TOKEN_LIFETIME"]`). Used for authenticating protected routes.
- **Refresh_token** : longer-lived (configured via `SIMPLE_JWT["REFRESH_TOKEN_LIFETIME"]`). Used to obtain new access tokens.
- Server sets cookies with attributes : `HttpOnly`, `Path=/`, `SameSite` per `JWT_COOKIE_SAMESITE` and `secure` per `JWT_COOKIE_SECURE`.
- Tokens include small custom claims: `role` , `tv` (token_version) , `pwd_ts` (optional).

Frontend integration tips (React)

1. Send login request with **credentials: "include"** so browser accepts `Set-Cookie` headers.
 2. On `200 OK`, store returned `user` object in app state and redirect to dashboard.
 3. For all subsequent protected requests, call fetch with **credentials: "include"**; the server will read the `access_token` cookie.
 4. When `401` due to expired access token, call the refresh endpoint `POST /api/auth/token/refresh/` (also **credentials: "include"**). On success refresh endpoint sets new cookies.
 5. Do **not** try to read token values from JavaScript (cookies are `HttpOnly`).
 6. Provide UI for unverified accounts (message + link/button to call `POST /api/auth/resend-verification/`).
-

Browser console fetch (paste into DevTools console)

```
fetch("http://localhost:8000/api/auth/login/", {
  method: "POST",
  credentials: "include",    // REQUIRED so browser stores HttpOnly cookies
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    email: "paras0mani6@example.com",
    password: "CounsellorPass1!"
  })
})
.then(async res => {
  const json = await res.json().catch(()=>({}));
  console.log("status:", res.status, json);
  if (res.status === 200) {
    // Save user in app state (example)
    window.currentUser = json.user;
  } else {
    // handle errors (show messages)
    alert(json.detail || "Login failed");
  }
})
.catch(err => console.error("Network error", err));
```

Using saved cookies for protected endpoint

```
// After successful login, call protected endpoint (must include credentials)
fetch("http://localhost:8000/api/auth/protected/", {
  method: "GET",
  credentials: "include"
})
.then(r => r.json().then(body => ({ status: r.status, body })))
.then(x => console.log(x))
.catch(e => console.error(e));
```