INTEROPERABILITY DEMONSTRATION

Sign in with ChatGPT

How the web app authenticates a ChatGPT account and sends a streaming GPT-5.5 prompt

Version 1.0 | July 17, 2026 | Local technical test utility

> **Purpose:** This small application demonstrates that ChatGPT OAuth and a subsequent model request can be implemented independently of OpenClaw or any other AI harness. It does not require an API key, a companion service, or transmission of a harness-specific originator identifier.

How Sign in with ChatGPT works

1. **Create a protected OAuth attempt.** The local server generates a PKCE verifier and SHA-256 challenge plus a random state value. The verifier remains on the local server.
2. **Open the authorization page.** The browser navigates to OpenAI's authorization service. The user signs in directly there; the demonstration app never receives the user's password.
3. **Return an authorization code.** After approval, the authorization service redirects to http://localhost:1455/auth/callback with a short-lived code and the original state value.
4. **Validate and exchange.** The local server verifies state, then exchanges the code and PKCE verifier for OAuth access and refresh credentials. This is a public-client PKCE flow and does not place a client secret in the browser.
5. **Establish the browser session.** The browser receives only an opaque HttpOnly session cookie. OAuth credentials remain server-side and are not exposed to page JavaScript or browser local storage.
6. **Refresh when required.** Before a model request, the server refreshes an expiring access credential and retries once after an authentication rejection when a refresh credential is available.

How a GPT-5.5 prompt is sent

- **Prompt submission.** The page sends the user's text to the local server over the same local web origin.
- **Authenticated model request.** The server retrieves the OAuth credential and ChatGPT account identifier from the vault, then sends a streaming request for model gpt-5.5.
- **Streaming response.** Text deltas are relayed to the browser with server-sent events, so the response appears progressively in the page.
- **Inspectable policy.** The page displays the actual application-generated header names recorded for the token exchange and model request.

Run the demonstration

- **Computer.** Extract the supplied .tgz archive, enter its package directory, run npm start, and open http://localhost:1455/. Then sign in, submit a prompt, and inspect the streamed response and request-policy panel.

No OpenClaw or harness dependency

Neither the sign-in sequence nor the model-request sequence starts, contacts, or depends on OpenClaw or another AI harness. The package contains no harness runtime, plugin, process, configuration, or account requirement.

- **Authorization URL.** No originator query parameter is included.
- **Token exchange.** No originator header and no application-supplied User-Agent header are included.
- **GPT-5.5 request.** No originator header and no application-supplied User-Agent header are included.
- **Account context.** The request carries the authenticated ChatGPT account context required for the account-scoped model request; this identifies the account, not an AI harness.

> **Browser limitation:** A normal browser always supplies its own browser User-Agent while navigating to the authorization website. Web page code cannot remove that protected browser header. It identifies the browser, not OpenClaw or another harness. The server-originated token and GPT requests omit User-Agent entirely.

Local credential storage

The synthetic Keychain substitute stores OAuth credentials in an AES-256-GCM encrypted vault under the app's local .data directory. A separate random 256-bit key and the encrypted vault file use owner-only file permissions. The portable package excludes the entire .data directory, so no account credential or test session is sent to another recipient.

This design keeps credentials out of browser JavaScript and plaintext files, but it is not equivalent to a hardware-backed operating-system Keychain: a local user or process that can read both vault files can decrypt them. The app is therefore intended as a local interoperability demonstration, not a public hosted service.

System requirements

Requirement	Details
Computer	macOS, Windows, or Linux on a platform supported by Node.js 20 or later. The package was tested on macOS.
Runtime	Node.js 20 or newer with npm. No npm install step or third-party JavaScript dependency is required.
Browser	A current Safari, Chrome, Edge, or Firefox release with cookies and pop-up navigation allowed for the local app.
Network	Outbound HTTPS access to auth.openai.com and chatgpt.com; local TCP port 1455 must be available and must not be exposed publicly.
Phone test	Optional: use the printed LAN address on the same network, then paste the final localhost callback address into the Phone callback URL field.
Account	A ChatGPT account and OAuth client accepted by the authorization service, with access to the requested GPT-5.5 model path.

Stopping and removing the demonstration

The package is portable. Normal use starts one foreground Node.js process and serves the site from the extracted folder. It does not install a daemon, LaunchAgent, scheduled task, browser extension, global npm package, or operating-system service.

Recommended removal sequence

1. **Sign out while the server is still running.** Use the page's Sign out button. This removes the access and refresh credentials from the local encrypted session record before shutdown.
2. **Stop the foreground server.** In the Terminal window that is running npm start, press Control-C. The local website immediately becomes unavailable because no separate background service exists.
3. **Recover a server whose Terminal window was lost.** Identify the process listening on local port 1455, then stop that process. Use the following commands.

macOS or Linux

```
lsof -nP -iTCP:1455 -sTCP:LISTEN
kill <PID>
```

Windows PowerShell

```
Get-NetTCPConnection -LocalPort 1455 -State Listen | Select-Object -ExpandProperty OwningProcess | ForEach-Object { Stop-Process -Id $_ }
```

4. **Delete the local files.** Delete the extracted package folder and, if it is no longer needed, the original .tgz archive. The package's .data folder is inside the extracted folder; deleting the folder therefore removes vault.key, sessions.enc.json, the static site, and all app-managed local session records.
5. **Optionally clear browser site data.** Remove stored site data for http://localhost:1455 and for any printed LAN address used from a phone. A remaining opaque session cookie cannot recover credentials after the server vault has been deleted, but clearing it completes browser-side cleanup.
6. **Confirm shutdown.** On macOS or Linux, run the lsof command below; no output means that nothing is listening on port 1455. On Windows, the corresponding Get-NetTCPConnection query should return no listening connection.

```
lsof -nP -iTCP:1455 -sTCP:LISTEN
```

> **OAuth scope of removal:** The Sign out action and folder deletion remove this app's local copies of the OAuth credentials. The demonstration does not call a remote OAuth-revocation endpoint, so a previously issued credential may remain server-valid until it expires or is revoked through the account provider. After the server is stopped and the vault is deleted, this installation has no credential with which to make another request.

What remains

Node.js remains installed because it is a general-purpose runtime, not part of this package. It may be left in place or removed separately using the same installer or package manager that originally installed it. No app-specific background process or website remains after the steps above.