# How single sign-on (SSO) with JWT for HelpCenter.io works?

**Build your help center**

## What JWT SSO is?

JWT SSO lets your users access your private HelpCenter.io help center – or use the embedded widget on your authenticated SaaS – **without seeing a login screen on our side**. Your application authenticates the user the way it already does. You then sign a short-lived JSON Web Token identifying who they are, and HelpCenter.io trusts that signed token.

There's no second password. No "Sign in with HelpCenter" button. No new account to provision in our dashboard.

## What it works on?

Three surfaces share the same JWT contract. You configure SSO once and it covers all three:

| Surface | What it looks like | How the JWT gets in |
| --- | --- | --- |

| | | |
|---|---|---|
| **Smart Widget** | The floating help button you embed inside your SaaS app via <script src=".../js/init.js"> | Host page passes the token via window.hcOptions.jwt; the widget sends Authorization: Bearer <token> on every API call |
| **JS-only help center** | The full help center rendered inside an iframe you embed on your own pages | Host page passes the token in the iframe URL: <iframe src="https://docs.example.com/?jwt=<token>"> |
| **Direct help center** | Visitors landing on your help center URL ( docs.example.com ) | The redirect-based flow described below — visitor is redirected to your login URL, your backend mints a JWT, redirects back to /sso/jwt?jwt=<token> |

You don't pick one — all three are gated by the same shared secret and the same set of required claims.

## The flow (direct help center)

1. A visitor lands on your private help center (e.g. https://docs.example.com/article/foo ).

2. We see no session and look up your SSO configuration.

3. We redirect the visitor to **your Login URL**, passing along where they were trying to go.

4. Your backend authenticates the visitor (or recognizes them from an existing session).

5. Your backend mints a JWT identifying the visitor, signed with the **shared secret** you and we agreed on. HS256.

6. Your backend 302s the visitor back to https://docs.example.com/sso/jwt?jwt=<token>.

7. We verify the signature, validate the claims, log the visitor in, and redirect them to the URL they originally requested.

## The flow (embedded widget)

1. Your host page authenticates the user the way it always has.

2. When the user lands on a page that embeds the widget, your backend mints a JWT for them (same shared secret, same claims).

3. The host snippet hands the JWT to the widget: window.hcOptions = { jwt: '<token>', ... }.

4. The widget includes Authorization: Bearer <token> on every call to our embed API.

5. We verify the signature on every request and apply your help center's visibility rules.

6. If the token expires while the widget is open, we return 403 SITE_AUTH_REQUIRED and the widget asks your onAuthExpired callback for a fresh token. No reload, no

login screen.

## What you're trusting us with

The shared secret. That's it. It signs every JWT you mint; we verify the signature against it. We never see your users' actual passwords; we never store them.

## Required claims

Every JWT must include these claims. If any are missing, the token is rejected with jwt_missing_required_claim.

- jti — unique token identifier. We use it for replay protection (the same jti cannot be used twice on the same site).
- iss — issuer. Typically your application's domain.
- iat — issued at, Unix seconds.
- exp — expiry, Unix seconds. **This is now required** (was previously optional). Keep it short — 5 minutes is typical.
- email — the user's email.
- name — the user's display name.

## Optional claims

- external_id — your stable user ID. Strongly recommended. Lets us track the same user across email changes.

- lang — two-letter language code. Defaults to the site's default language.
- avatar_url — public URL to the user's avatar.
- role — viewer, editor, or admin. Defaults to viewer.
- custom_fields — free-form object surfaced on the User record.
- aud — audience. Only required if you've set an Audience value in our dashboard.

## What you control from the dashboard

**Settings → Security → SSO** is where this all lives. Self-serve, no support ticket needed (was Enterprise-only with a support handoff prior to v2.18 — that changed).

- **Login URL** — where to send visitors to be authenticated.
- **Shared Secret** — generate one or paste your own. Minimum 64 characters.
- **Logout URL** — optional. Where to send visitors after logout.
- **Issuer** — optional. If set, JWTs whose iss doesn't match are rejected.
- **Audience** — optional. Same idea for aud.
- **Token TTL** — hard cap on token age applied by the widget surface, even if exp is generous. Defaults to 300 seconds.

## Plan availability

JWT SSO is available on our **Catalyst, and Enterprise** plans. No support ticket required to enable. You can open the Settings page and configure it.

Other SSO providers (SAML 2.0, OIDC, Google Workspace, Okta, Microsoft Entra ID) are on the roadmap. Our special placeholder cards in your dashboard let you signal demand for the one you want most.

## Security notes

- Tokens are **single-use** on the widget surface. A replayed jti is rejected. Mint a fresh token for each session your host page starts.
- The widget never stores the JWT in localStorage. It lives in memory only for the lifetime of the page.
- When the host snippet passes the JWT to the widget iframe, it does so via the URL **hash** (#jwt=...), not the query string. Hashes are not sent to the server — the token doesn't land in proxy or access logs.
- Clock skew tolerance is 30 seconds on both iat and exp. Sync your servers via NTP.