

Authentication & API Endpoints Guide

Generated on November 1, 2026

Authentication & API Endpoints Guide

How your application signs users in to your support portal, ends their sessions, and verifies members. Always use HTTPS.

Your portal is reachable on your own ticket domain (for example `https://ticket.yourtenant.com`). The examples below use that form.

All server-to-server calls authenticate with your **API Key** and **API Secret** from **Settings** › **API**, sent as the `X-API-Key` and `X-API-Secret` headers. Keep the secret on your server: never put it in browser JavaScript or mobile apps.

1) Single sign-on with a redirect token

Your server asks the ticket system for a short-lived sign-in link for the current user, then redirects the user's browser to it.

- Endpoint: `POST /api/auth/redirect`
- Authentication: `X-API-Key` + `X-API-Secret`
- Body: `tenant` (your portal's slug), `email`, `name`, and optionally `redirect_url` (a page in your portal to open after sign-in, e.g. `/tickets/42`)

```
curl -X POST "https://ticket.yourtenant.com/api/auth/redirect" \
  -H "Content-Type: application/json" \
  -H "X-API-Key: tk_your_api_key" \
  -H "X-API-Secret: your_api_secret" \
  -d '{"tenant": "yourtenant", "email": "user@yourtenant.com", "name": "Jane Doe", "redirect_url": "/tickets/42"}'
```

The response redirects to a one-time sign-in link of the form `https://ticket.yourtenant.com/auth/callback?token=...`. Send the user's browser there. The link expires after a few minutes.

2) Single sign-on with a JWT signed by your server

Your server signs a short-lived JWT with your **SSO secret** (Settings > API) and posts it to the ticket system.

- Endpoint: `POST /api/jwt/login`
- Body: `token` (the signed JWT) and `tenant_slug` (your portal's slug)
- Token format: a standard HS256 JWT (`header.payload.signature`). Use a standard JWT library.
- Payload:
 - `email` (string) — the user's email
 - `name` (string) — the user's full name
 - `iat` (int) — issued at (Unix time)
 - `exp` (int) — expiry (Unix time); keep it short, e.g. `iat + 120`
 - `jti` (string) — a unique id per token, so a token cannot be reused

```
curl -X POST "https://ticket.yourtenant.com/api/jwt/login" \  
-d "token=<SIGNED_JWT>" \  
-d "tenant_slug=yourtenant"
```

A successful response contains the user and a `redirect_url` to open in the user's browser.

3) Ticket API

Creating, listing and updating tickets uses API v2 under `/api/v2/*` with `X-API-Key` + `X-API-Secret`. See the **API Integration Guide** for every endpoint, request and response.

4) Remote logout

When a customer logs out of your application, tell the ticket system so their support session ends too.

- Endpoint: `POST /api/auth/logout`
- Authentication: `X-API-Key` + `X-API-Secret`
- Body: `email` or `external_user_id` (at least one)

```
curl -X POST "https://ticket.yourtenant.com/api/auth/logout" \
-H "Content-Type: application/json" \
-H "X-API-Key: tk_your_api_key" \
-H "X-API-Secret: your_api_secret" \
-d '{"email": "user@yourtenant.com"}'
```

The response is always `200` with `{ "success": true, "message": "Logout processed" }`, whether or not the user exists. The customer is signed out of the support portal on their next page load.

5) Member verification webhook (an endpoint you build)

This is **not** an endpoint on the ticket system — it is an endpoint **you implement** in your application. The ticket system calls it when a staff member files a ticket for someone who does not have a support account yet, to confirm that the email belongs to one of your members.

Configure it

Ask your Central Tickets administrator to set:

Field	Value
Verification URL	<code>https://yourapp.com/api/verify-member</code>
Verification token	a long random secret that only you and the ticket system know

Request the ticket system sends

```
POST {verification URL}
Authorization: Bearer {verification token}
Content-Type: application/json
Accept: application/json

{ "email": "user@example.com" }
```

The call is made server-to-server with a 10-second timeout.

Response your endpoint must return

User exists (**200 OK**):

```
{ "exists": true, "user": { "name": "Jane Doe", "email": "user@example.com" } }
```

User does not exist (**200 OK**):

```
{ "exists": false }
```

Any non-2xx response is treated as an error, and the staff member sees "unable to verify".

What happens next

Result	Outcome
The user already has a support account	Ticket created, no call made
No verification URL configured	Staff are told the user must already be registered
Your endpoint returns exists: false	Staff are told the user is not a member
Your endpoint returns exists: true	A support account is created, the ticket is filed, and the user is emailed a link to it
Your endpoint is unreachable or returns non-2xx	Staff see an error; no ticket is created

Security requirements for your endpoint

1. Check that the `Authorization: Bearer` token matches your verification token; reject anything else with `401`.
 2. Return only `name` and `email` — nothing else about the user.
 3. Accept server-to-server calls only; never expose it to browsers.
-

Good practice

- Keep every token short-lived and include a unique `jti` in JWTs.
- Store your API secret, SSO secret and verification token in server-side configuration, never in source code or front-end code.
- If a secret may have leaked, rotate it in **Settings › API** straight away.