Authentication & API Endpoints Guide

Generated on August 31, 2026

Tenant API Endpoints — Integration Reference

This document lists the HTTP endpoints tenant applications (external apps) call to integrate with the ticket system. It covers both SSO redirect tokens and the JWT login flow, required parameters, authentication expectations, and quick curl examples.

Important: the platform supports two routing modes:

- Subdomain mode (production): tenant resolved from Host header (ticket.). Use this when your app can call from the tenant subdomain.
- Slug/path mode (local/dev): tenant is passed in the URL as /{tenant_slug}/... — use when testing locally or when Host cannot be trusted.

Always use HTTPS in staging/production.

---

1) SSO Redirect (encrypted token)

Purpose: central platform issues an encrypted redirect token which the user browser is redirected to, allowing the platform to complete SSO flow.

- Endpoint to redirect users TO (platform callback):
  - Subdomain: `GET https://ticket.<tenant-domain>/auth/callback?token=<ENCRYPTED_TOKEN>`
  - Slug (dev/local): `GET https://<host>/<tenant_slug>/auth/callback?token=<ENCRYPTED_TOKEN>`
- How it's produced: tenant (or central master) obtains an encrypted token via the platform's `AuthRedirectService` (server-side) and issues a 302 redirect to the callback URL above.

- Request fields: `token` — encrypted payload (handled by platform services). The callback will validate, create/find user, and create a session.

Quick example (platform-side redirect):

```
Location: https://ticket.yourtenant.com/auth/callback?token=<ENCRYPTED_TOKEN>
```

Notes:

- This flow is browser-based (redirect). The token is encrypted by the platform; external apps typically request the platform to generate it rather than generate it themselves.

---

2) JWT Login (external app issues signed token)

Purpose: external tenant app generates a short-lived HMAC-SHA256-signed token that the platform accepts to create a session for a user.

- Endpoint: `GET` or `POST` to:
  - Subdomain: `https://ticket.<tenant-domain>/api/jwt/login`
  - Slug (local/dev): `https://<host>/<tenant_slug>/api/jwt/login`

- Required parameters (query or form):
  - `token` (string) — the signed token. Supports two formats:
    1. **Standard JWT**: `header.payload.signature` (3-parts, standard libraries)
    2. **Custom JWT**: `base64(payload).signature` (2-parts, legacy)
  - `tenant_slug` (string) — tenant slug (required unless host/subdomain resolution guaranteed)
  - `intended` (string, optional) — URL to redirect user to after session created (e.g. `/tickets/123`)

- Token payload requirements (JSON encoded then base64):

    - `email` (string) — user email
    - `name` (string) — full name
    - `iat` (int) — issued at (unix timestamp)
    - `exp` (int) — expiry (unix timestamp; short TTL recommended, e.g. iat + 120)
    - `jti` (string, recommended) — unique ID for replay protection

- Auth: token itself authenticates the user flow; server-to-server API calls use `X-Tenant-API-Token` or v2 `X-API-Key/X-API-Secret` as appropriate.

Example (curl GET redirect-style):

```
curl -i "https://ticket.yourtenant.com/api/jwt/login?token=<TOKEN>&tenant_slug=yourtenant&inter
```

Example (curl POST):

```
curl -X POST "https://ticket.yourtenant.com/api/jwt/login" \
  -d "token=<TOKEN>" \
  -d "tenant_slug=yourtenant" \
  -d "intended=/tickets/5"
```

Notes:

- If your app runs from the tenant subdomain and Host is trustworthy, the platform may resolve tenant from Host and `tenant_slug` can be omitted — but passing `tenant_slug` is recommended for determinism.
- Platform verifies signature with the tenant's `jwt_secret` (Blue Banner in Settings) and checks `iat` / `exp` and JTI replay cache.
- **Note**: Both standard 3-part JWTs and legacy 2-part tokens are supported. Standard JWTs are recommended for new integrations.

---

3) API endpoints tenants call for normal operations (server-to-server / API)

These are tenant-scoped API endpoints (use tenant API token or JWT-based auth depending on your integration):

- POST `/api/tickets` — Create a ticket. Body: `title`, `description` (or `description` becomes first reply), `category_id`, etc.
- GET `/api/tickets` — List tickets for tenant
- GET `/api/tickets/{id}` — Get ticket
- PUT `/api/tickets/{id}` — Update ticket
- DELETE `/api/tickets/{id}` — Delete ticket
- GET `/api/categories` — List categories
- GET `/api/tenant` — Return tenant info (configuration)

Authentication methods for these APIs:

- `X-Tenant-API-Token` header — tenant API token (server-to-server)
- Bearer token — platform-issued session JWT (user flows)
- v2 APIs under `/api/v2/*` use `X-API-Key` + `X-API-Secret` (Amber Banner in Settings). See `AuthenticateTenantApiV2` middleware.

---

4) SSO/token helper endpoints

- `POST /api/auth/redirect` — Request platform to generate an encrypted redirect token and return a 302 to the platform callback. Use when the external system asks the platform to generate the encrypted redirect token. The body must include the `tenant` slug.
- `GET /api/auth/callback` — Platform endpoint that processes encrypted redirect tokens (used in redirect flow described in section 1).
- `POST /api/sso/token` and `POST /api/sso/verify` — Supporting endpoints used by SSO workflows (internal or partner integrations).

---

5) Remote Logout (Push-Based Session Sync)

Purpose: When a user logs out of the tenant application, notify the ticket system so the user's ticket session is also ended on their next request.

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

- Authentication: **API v2 credentials** — `X-API-Key` + `X-API-Secret` headers (from **Settings > API**)

Request body (one of):

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

```
{ "external_user_id": "12345" }
```

At least one of `email` or `external_user_id` must be provided.

Quick example (curl):

```
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"}'
```

Response (always 200 to prevent user enumeration):

```
{ "success": true, "message": "Logout processed" }
```

How it works:

1. Tenant app calls `POST /api/auth/logout` with the user's email
2. Ticket system sets a cache flag: `force_logout:{tenant_id}:{user_id}`
3. On the user's next request, the `VerifyTenantSessionSync` middleware detects the flag
4. User is logged out, session invalidated, and redirected to the tenant login page

Notes:

- The endpoint does **not** log the user out immediately — it sets a flag that the middleware picks up on the next request
- The flag TTL is `session.lifetime + 5 minutes` (auto-cleanup)
- Only users with role `user` are affected (admins and agents are never force-logged-out)

- This replaces the old polling-based approach (`ExternalAuthSessionService`) which has been removed

---

6) v2 API notes

- v2 routes live under `/api/v2/*` and require `X-API-Key` + `X-API-Secret` headers for authentication. Use these for programmatic tenant operations when available.

---

7) Member Verification Webhook (tenant-side endpoint)

This is **not** an endpoint on the ticket system — it is an endpoint **you must implement** in your application. The ticket system calls it when a staff member uses the **"File Ticket for User"** feature to create a ticket for someone who doesn't yet have a local account. The resulting ticket is labelled **"Filed for [user]"** in the conversation thread.

Purpose: Confirm that a given email address belongs to a registered member of the tenant application before provisioning them and creating a ticket for them.

Configure it

Set both the URL and verification token in the master Tenant edit form:

Field	Value
`user_verification_url`	`https://yourapp.com/api/verify-member`
`verification_token`	`your_shared_secret_here`

The field accepts any HTTPS URL. It is called synchronously at ticket creation time (10 second timeout).

Request the ticket system sends

```
POST {user_verification_url}
Authorization: Bearer {tenant.verification_token}
Content-Type: application/json
Accept: application/json

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

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
}
```

> *Always return `200 OK`. Non-2xx responses are treated as errors and presented to the staff member as "unable to verify".*

How the ticket system responds to the result

Scenario	Outcome
User exists in ticket system locally	Ticket created immediately, no external call made
User not local, `user_verification_url` not set	Error: staff is informed the user must already be registered
User not local, external returns `exists: false`	Error: "This user is not a member of the tenant application"
User not local, external returns `exists: true`	User auto-provisioned (role `user`, random password), ticket created, email sent to user
External endpoint unreachable or returns non-2xx	Error surfaced to staff member, ticket not created

Email notification to the user

Once a ticket is filed for a user, the ticket system sends them an email with:

- Who created the ticket on their behalf
- Ticket title, description, priority, and category
- A direct link to view and respond to the ticket

The email uses the tenant's configured SMTP settings (or falls back to global settings).

Security requirements for your endpoint

1. Validate the `Authorization: Bearer` token matches your configured verification secret
2. Return `401` or any non-2xx status to reject unauthorized requests
3. Do not leak sensitive user data — only `name` and `email` are needed in the response
4. The endpoint is called server-to-server only, never from the browser

Quick example (curl, for testing):

```
curl -X POST "https://yourapp.com/api/verify-member" \
  -H "Authorization: Bearer your_shared_verification_secret" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -d '{"email": "user@example.com"}'
```

Best practices and debugging tips

- Always use HTTPS in staging/production. For local testing you can use slug-mode (path-based) to avoid subdomain setup.
- Keep token TTL short (2 minutes recommended) and include `jti` to protect against replay.
- Provide `tenant_slug` with calls unless you can guarantee call originates from the tenant subdomain.
- Check platform logs for verification failures: signature mismatch, expired, missing claims, tenant not found.
- When testing locally, use `php artisan serve` with slug path: `http://127.0.0.1:8000/{tenant_slug}/api/jwt/login`.

If you want, I can also:

- generate an OpenAPI fragment for these endpoints, or
- add a small `docs/examples/jwt_login.sh` script with token generation + curl examples for local testing.

File: `docs/TENANT_API_ENDPOINTS.md`