ޓެކްސް އެޕީލް ޓްރައިބިއުނަލް
މާލެ،
ދިވެހިރާއްޖެ.

Tax Appeal Tribunal

ANNEX-1

TAX APPEAL TRIBUNAL

Case Management System

API & System Architecture Specification

Version 1.0 | Document prepared: 13 September 2026

Companion document to: TAT Case Management System - Full System Documentation & Implementation Timeline (v1.4)

1. Introduction

1.1 Purpose

This document specifies the Application Programming Interface (API) and system architecture for the Tax Appeal Tribunal (TAT) Case Management System (CMS). It extends the functional requirements set out in "TAT Case Management System - Full System Documentation" (v1.4) with a concrete, API-first technical design: a backend API service that is fully decoupled from the frontend application, built so that the same set of APIs can be consumed by the TAT frontend, other government organizations (e.g. the Maldives Inland Revenue Authority - MIRA), and third-party applications operating under a data-sharing agreement with TAT.

1.2 Design Principles

- API-first: every capability of the CMS - registration, case timeline, meetings, hearings, reports, notifications - is exposed as a documented REST API. The frontend is simply the first consumer of that API, not a privileged one.
- Separation of concerns: the backend (API + database + business logic) and the frontend (browser application) are two independently deployable projects, in two separate source repositories, communicating only over HTTPS/JSON.
- External-consumer ready: authentication, authorization, versioning, rate limiting and documentation are designed from day one for consumers outside TAT's own frontend - not retrofitted later.
- Least privilege: internal (TAT staff) and external (government/partner) consumers are issued different credentials, scopes and rate limits, enforced by an API gateway in front of the backend.
- Everything documented: every endpoint, every non-framework technology, and every integration point is documented in this specification or in an auto-generated OpenAPI contract, so a new government consumer can integrate without needing to read source code.

1.3 Scope

This document covers: the recommended technology stack (Section 2), the API design standards all endpoints must follow (Section 3), the authentication and authorization model for internal and external consumers (Section 4), the full catalogue of API endpoints grouped by CMS module, one line per endpoint (Section 5), the external Partner API surface for other government organizations and apps (Section 6), how the APIs are built and documented in practice (Section 7), and documentation requirements for any technology used outside the core frameworks (Section 8).

2. Technology Stack

The stack below uses current, actively-maintained major versions as of the 2026 build window referenced in the implementation timeline (Phases 2-11, October 2026 - December 2026). Exact patch versions should be pinned by the delivery team at the start of Phase 3 (Environment & Architecture Setup) to whatever is the latest stable release at that time.

2.1 Backend - Vendor-Choice Framework Options

To avoid excluding qualified bidders by mandating a single programming language, TAT accepts three backend stack options for this tender. A bidding vendor selects ONE option and builds the entire backend in it - options are not to be mixed within a single deployment. Whichever option is chosen, the backend MUST implement the identical API contract defined in Sections 3, 5 and 6 of this document (same routes, request/response shapes, auth model and OpenAPI documentation requirement) - the framework choice affects only how the contract is built, never what the contract is.

Option A - ASP.NET Core (C#)

Layer	Technology
Runtime / Framework	.NET 9 + ASP.NET Core Web API (C# 13)
ORM / Data Access	Entity Framework Core 9
Background Jobs	Hangfire
Real-time updates	SignalR
API Documentation	Swashbuckle / NSwag -> OpenAPI 3.1
Testing	xUnit, WebApplicationFactory, Testcontainers (Postgres)

Long-term Microsoft-supported stack widely used in government systems; first-class OpenAPI generation, strong typing, high throughput, built-in dependency injection.

Option B - Laravel (PHP)

Layer	Technology
Runtime / Framework	PHP 8.3 + Laravel 11
ORM / Data Access	Eloquent ORM + Laravel migrations
Background Jobs	Laravel Queues (Redis/database driver) + Laravel Scheduler for the 24-hour due-date job
Real-time updates	Laravel Reverb (WebSockets) or Laravel Echo + Pusher-protocol broadcasting
API Documentation	L5-Swagger (Swagger-PHP) or Scribe -> OpenAPI 3.1
Testing	Pest or PHPUnit, with a disposable Postgres test database per run

Mature, widely-available developer pool; Eloquent and Artisan give fast, convention-driven CRUD scaffolding well suited to the many reference-data and case-entity endpoints in Section 5.

Option C - Django (Python)

Layer	Technology
Runtime / Framework	Python 3.12 + Django 5 + Django REST Framework
ORM / Data Access	Django ORM + Django migrations
Background Jobs	Celery + Redis/RabbitMQ broker, Celery Beat for the 24-hour due-date job
Real-time updates	Django Channels (ASGI, WebSockets)
API Documentation	drf-spectacular -> OpenAPI 3.1
Testing	Django test runner / pytest-django, with a disposable Postgres test database per run

Batteries-included admin and ORM speed up the Admin and Reference Data modules (Section 5.2-5.5); the same language as the OCR microservice (Section 8) lets a Django-choosing vendor build OCR as an in-process app instead of a separate service.

Shared components (all options)

Layer	Technology	Why
Database	PostgreSQL 16	Open-source, no per-core licensing (important for a government budget), strong JSON/full-text support for case search, mature backup/replication tooling; supported by all three ORMs above.
Identity Provider	Keycloak (latest LTS)	Open-source OpenID Connect / OAuth 2.0 server, self-hostable on TAT infrastructure for data sovereignty; issues standard JWTs that any of the three frameworks validates the same way (Section 4).
API Gateway	Kong Gateway (or YARP if Option A is chosen)	Single ingress point for rate limiting, API-key/token validation, request logging and routing between the internal API and the external Partner API surface (Section 6), independent of the backend framework.
Containerisation	Docker + Docker Compose (dev), Kubernetes or on-prem container platform (prod)	Every option above ships an official Docker base image, so environments (Phase 3 of the timeline) are consistent regardless of which option is selected.
1.3	14 Sep 2026	Implementation timeline reference (Section 2.1) updated from Phases 2-11 October 2026 - January 2027 to October 2026 - December 2026, reflecting the compressed build schedule now targeting a 31 December 2026 go-live.

2.2 Frontend - Vendor-Choice Framework Options

As with the backend (Section 2.1), TAT accepts more than one frontend option so bidders are not excluded by front-end technology preference. A vendor selects ONE option for the whole frontend. Two of the three options are backend-agnostic SPAs; the third (Livewire) is a special case that changes the deployment model and is only available if Laravel (Option B) is also the chosen backend - see the compatibility note after Option 3.

Option 1 - React (decoupled SPA)

Layer	Technology
Framework	React 19 + TypeScript 5
Build tool	Vite 6
API data layer	TanStack Query
UI components	Tailwind CSS + shadcn/ui
Tables / lists	TanStack Table
Charts	Recharts
Forms & validation	React Hook Form + Zod
Auth in the browser	oidc-client-ts (Authorization Code + PKCE against Keycloak)

Current, widely-supported SPA framework; TypeScript gives compile-time safety against the many typed fields in the case model (dates, enums, status codes). Works against any backend option (A, B or C) purely through the REST contract in Section 5 - no coupling to backend language.

Option 2 - Vue (decoupled SPA)

Layer	Technology
Framework	Vue 3 (Composition API) + TypeScript 5
Build tool	Vite 6
API data layer	TanStack Query (Vue adapter) or Pinia + a typed fetch client
UI components	Tailwind CSS + PrimeVue or shadcn-vue
Tables / lists	TanStack Table (Vue adapter)
Charts	vue-chartjs or ECharts for Vue
Forms & validation	VeeValidate + Zod
Auth in the browser	oidc-client-ts (framework-agnostic; Authorization Code + PKCE against Keycloak)

Equally current SPA framework with a gentler learning curve for teams already comfortable with Blade-style templating; same compile-time typing benefits as Option 1. Also works against any backend option (A, B or C) purely through the REST contract in Section 5.

Option 3 - Livewire (Laravel-coupled, server-rendered)

Layer	Technology
Framework	Laravel 11 + Livewire 3 + Alpine.js, Blade templates
UI components	Tailwind CSS + a Blade/Livewire component kit (e.g. Flux or Livewire UI)
Charts	Chart.js via a Livewire/Alpine wrapper component
Forms & validation	Laravel's own Form Request validation, reused server-side (no separate client validation layer)
Auth in the browser	Laravel's session-based auth via Laravel Socialite/OIDC client against Keycloak

IMPORTANT - architectural exception: Livewire components render server-side inside the Laravel application and talk to it over Livewire's own internal AJAX/WebSocket protocol, not over the public REST API in Sections 3/5/6. Selecting Livewire therefore has two consequences that do not apply to Options 1 and 2:

- Backend dependency: Livewire is a Laravel package, so Option 3 is only available when Option B (Laravel) is also the selected backend (Section 2.1) - it cannot be paired with Option A (ASP.NET Core) or Option C (Django).
- Deployment coupling: the frontend and backend become one deployable Laravel application (one repository, one release cadence for the staff-facing UI), which is an explicit exception to the "two independently deployable projects" principle in Section 1.2. This is acceptable ONLY because the same Laravel backend must still separately expose the full, unchanged REST API of Sections 5 and 6 for the frontend to optionally also use and for every other consumer (government organizations, partner apps, SMS/OCR integrations) required by Section 1.1 - Livewire is an additional internal UI layer on top of that API, never a replacement for it.

2.3 Why an API-first, decoupled architecture

- The same backend serves three audiences without three implementations: the TAT staff frontend, other government organizations (Section 6), and any future channel (mobile app, public status-check kiosk) - each simply becomes another authenticated API consumer.
- The frontend can be replaced, redesigned or rebuilt (e.g. a future mobile app for Bench Members) without touching the backend or re-negotiating data access with government partners.
- External integrations (MIRA, SMS gateway, OCR) integrate against the same versioned, documented contract as the frontend, so they get the same guarantees of stability and backward compatibility.
- Because the frontend and any government/partner consumer talk to the backend only through the OpenAPI contract in Sections 3, 5 and 6, the backend's internal framework choice (Section 2.1, Option A/B/C) is invisible to every consumer - it can even be changed in a future re-procurement without breaking existing integrations, provided the contract is preserved.
- If the React or Vue frontend option is chosen (Section 2.2, Option 1 or 2), the frontend is a standalone single-page application, built and deployed independently of the backend (its own repository, its own CI pipeline, its own release cadence), with no direct database access and no shared code with the backend beyond the published OpenAPI contract, from which its TypeScript API client is generated. If Livewire (Option 3) is chosen instead, this separation applies only between the Laravel/Livewire application and every external consumer - not between "frontend" and "backend" internally - per the exception noted under Section 2.2, Option 3.

3. API Design Standards

Every endpoint listed in Sections 5 and 6 - without exception - must conform to the standards below.

3.1 Protocol & format

- REST over HTTPS (TLS 1.3) only; HTTP is not served, including internally.
- JSON request/response bodies (application/json); dates in ISO 8601 (UTC); Case Number format exactly as defined in the requirements: TAT-CA-(case type)/(registered year)/(auto-generated number).
- Resource-oriented URLs, plural nouns, nesting reflects ownership, e.g. /api/v1/cases/{caseId}/hearings/{hearingId}.

3.2 Versioning

- URI versioning: /api/v1/..., /partner/v1/... A breaking change ships as v2 alongside v1, with v1 supported for a published deprecation window (minimum 12 months for external/partner endpoints, reflecting the long-lived nature of government integrations).

3.3 Pagination, filtering, sorting

- List endpoints accept ?page=&pageSize= (default pageSize 25, max 200) and return {data, page, pageSize, totalCount, totalPages}.
- Filtering via query parameters matching the field name, e.g. ?status=HearingPending&taxType=G&year=2027.
- Sorting via ?sort=fieldName or ?sort=-fieldName for descending.

3.4 Errors

- All errors use the RFC 7807 "Problem Details" JSON format: {type, title, status, detail, traceId, errors[]} so every consumer - internal or external - parses errors the same way.
- Validation errors return HTTP 422 with a field-level errors array; authorization failures return 401 (no/invalid token) or 403 (valid token, insufficient scope/role).

3.5 Idempotency & concurrency

- State-changing POST requests that must not be duplicated (e.g. Register Case, Send SMS) require an Idempotency-Key header; a repeated key returns the original result instead of creating a duplicate.
- Updates use optimistic concurrency via an ETag / If-Match header to prevent two staff members silently overwriting each other's changes to the same case.

3.6 Rate limiting & throttling

- Enforced at the API gateway (Section 2.1), not in application code, so limits differ by consumer tier: internal frontend (high limit, effectively unrestricted for authenticated staff), government

partner (agreed quota per data-sharing agreement, e.g. 60 requests/minute), unauthenticated (0 - no anonymous access).

- Throttled requests return 429 Too Many Requests with a Retry-After header.

3.7 Auditability

- Every API call that creates, updates or deletes data is written to the audit log (Section 5.16) with: timestamp, calling user or client, endpoint, entity affected, before/after values for changed fields. This directly fulfils requirement Section 3.9 ("All logins, changes should be recorded") of the Full Documentation.

3.8 Data protection

- Personally identifiable fields (ID Card No, TIN No, phone, email, addresses for Appellant, representative and lawyer) are encrypted at rest (column-level encryption) and only ever transmitted over TLS.
- Partner API responses (Section 6) return the minimum data necessary for the partner's purpose - full PII is never returned to a partner unless that field is explicitly within the data-sharing agreement for that partner's scope.

4. Authentication & Authorization Model

No specific national identity/gateway standard has been mandated for TAT at the time of writing. The model below follows common, vendor-neutral government API practice (OAuth 2.0 / OpenID Connect, as used by most national government API gateways) and should be confirmed with TAT/NCIT (or equivalent national IT authority) during Phase 2 (Requirements Sign-off & Solution Design).

4.1 Identity Provider

A single Keycloak instance, operated by TAT, hosts two realms:

Realm	Consumers	Flow
tat-internal	TAT staff using the CMS frontend (Administrator, Supervisor, User roles; functional roles Bench Chair, Bench Member, Case Analyst, CRT/SLO/TO, LSH, Chairperson mapped as claims)	Authorization Code + PKCE (browser login), short-lived access token (15 min) + refresh token (8 hr, sliding)
tat-partners	Other government organizations and registered third-party apps (Section 6)	Client Credentials (machine-to-machine); access token (60 min), no refresh token - re-request on expiry

4.2 Roles and scopes

Role / Scope	Applies to	Grants
Administrator	Internal	Full access: users, access levels, all reference data, all cases.
Supervisor	Internal	Case assignment (Bench, Case Analyst, SLO/TO), oversight views, no user/access-level management.
User	Internal	Operational access to assigned cases, meetings, hearings, reviews.
cases:read	Partner	Read-only access to the Partner case-status/summary endpoints (Section 6).
cases:write:mira	Partner (MIRA only)	Submit MIRA Response / MIRA Statement documents into a case (Section 6).
lookups:read	Partner	Read-only access to public reference lists (case types, tax types) for form validation.
webhooks:manage	Partner	Register/remove webhook subscriptions for case status-change events.

4.3 Partner onboarding

1. A government organization or approved app requests access; TAT reviews and signs a data-sharing agreement defining which scopes and rate limit apply.
2. An Administrator registers the partner as a client in the tat-partners realm and issues a client ID/secret (or mTLS certificate for higher-sensitivity partners such as MIRA).
3. The partner calls POST /partner/v1/oauth/token with its client credentials to obtain an access token, then calls the Partner APIs in Section 6 with Authorization: Bearer <token>.
4. All partner traffic passes through the API gateway (Section 2.1), which enforces the agreed rate limit and logs every call for audit (Section 3.7).

4.4 Machine-to-machine integrations (SMS, OCR)

The SMS gateway and OCR service (Section 3.10 of the Full Documentation) are outbound integrations called by the CMS backend, not inbound consumers. Their credentials (API keys) are stored in a secrets manager (e.g. Azure Key Vault / HashiCorp Vault), never in source control, and are scoped to the single backend service account that calls them (Section 5.19).

5. Internal API Catalogue

All endpoints below are served under the base path /api/v1 and require a valid tat-internal access token (Section 4.1) unless stated otherwise. Each row is one API. "Access" lists the minimum role required, per Section 4.2, and/or the functional role (e.g. TO, SLO) referenced in the source workflow documents.

5.1 Authentication & Session

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/auth/login	Exchange TAT credentials for an authorization code via Keycloak (redirect flow entry point).	Public
POST	/api/v1/auth/token	Exchange an authorization code for an access + refresh token.	Public
POST	/api/v1/auth/refresh	Exchange a refresh token for a new access token.	Authenticated
POST	/api/v1/auth/logout	Revoke the current session and refresh token.	Authenticated
GET	/api/v1/auth/me	Return the authenticated user's profile, access level and functional roles.	Authenticated

5.2 Admin - Users & Access Levels

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/admin/users	List users (paginated, filter by access level).	Administrator
POST	/api/v1/admin/users	Create a user: Name, Email address, Access level.	Administrator
GET	/api/v1/admin/users/{userId}	Get a single user's details.	Administrator
PATCH	/api/v1/admin/users/{userId}	Amend a user's Name/Email.	Administrator
DELETE	/api/v1/admin/users/{userId}	Delete/deactivate a user.	Administrator
PATCH	/api/v1/admin/users/{userId}/access-level	Change a user's access level; records the change date.	Administrator
GET	/api/v1/admin/access-levels	List access levels (Administrator / Supervisor / User).	Administrator
POST	/api/v1/admin/access-levels	Create a new access level.	Administrator
PATCH	/api/v1/admin/access-levels/{id}	Change an access level's permissions.	Administrator
DELETE	/api/v1/admin/access-levels/{id}	Delete an access level.	Administrator

5.3 Admin - Members List

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/admin/members	List Tribunal members (Name, Designation, Email).	Administrator
POST	/api/v1/admin/members	Add a member.	Administrator
GET	/api/v1/admin/members/{memberId}	Get a member's details.	Administrator
PATCH	/api/v1/admin/members/{memberId}	Amend a member's details.	Administrator
DELETE	/api/v1/admin/members/{memberId}	Remove a member.	Administrator

5.4 Admin - Staff List

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/admin/staff	List staff (Name, Designation, Email).	Administrator
POST	/api/v1/admin/staff	Add a staff member.	Administrator
GET	/api/v1/admin/staff/{staffId}	Get a staff member's details.	Administrator
PATCH	/api/v1/admin/staff/{staffId}	Amend a staff member's details.	Administrator
DELETE	/api/v1/admin/staff/{staffId}	Remove a staff member.	Administrator

5.5 Reference Data

The same five operations apply to each of the six configurable lists below; every row is a distinct endpoint.

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/reference/case-types	List Type of Cases (Tax Amount, Fine, Objected to submit documents, Extension of Audit duration, Objection Rejection, Decision to collect third party taxes, Civil Action, Other tax related).	Authenticated
POST	/api/v1/reference/case-types	Add a new entry to case-types.	Administrator
PATCH	/api/v1/reference/case-types/{id}	Amend an entry in case-types.	Administrator
DELETE	/api/v1/reference/case-types/{id}	Delete an entry from case-types.	Administrator
GET	/api/v1/reference/tax-types	List Type of tax (BPT, GST, WHT, GT, RT, Other).	Authenticated
POST	/api/v1/reference/tax-types	Add a new entry to tax-types.	Administrator
PATCH	/api/v1/reference/tax-types/{id}	Amend an entry in tax-types.	Administrator
DELETE	/api/v1/reference/tax-types/{id}	Delete an entry from tax-types.	Administrator
GET	/api/v1/reference/rejection-reasons	List Reason of Rejection (Non-Compliance with Law, Not Paid 25%, Out of TAT Mandate).	Authenticated
POST	/api/v1/reference/rejection-reasons	Add a new entry to rejection-reasons.	Administrator
PATCH	/api/v1/reference/rejection-reasons/{id}	Amend an entry in rejection-reasons.	Administrator
DELETE	/api/v1/reference/rejection-reasons/{id}	Delete an entry from rejection-reasons.	Administrator
GET	/api/v1/reference/phases	List Type of Phases (Case Meetings, Hearings, Decision Drafting, Legal Review, Member's Review, Decision Finalizing).	Authenticated
POST	/api/v1/reference/phases	Add a new entry to phases.	Administrator
PATCH	/api/v1/reference/phases/{id}	Amend an entry in phases.	Administrator
DELETE	/api/v1/reference/phases/{id}	Delete an entry from phases.	Administrator
GET	/api/v1/reference/meeting-types	List Meeting Types (Pre-hearing, Pre-decision, Case Discussion).	Authenticated
POST	/api/v1/reference/meeting-types	Add a new entry to meeting-types.	Administrator
PATCH	/api/v1/reference/meeting-types/{id}	Amend an entry in meeting-types.	Administrator
DELETE	/api/v1/reference/meeting-types/{id}	Delete an entry from meeting-types.	Administrator

GET	/api/v1/reference/hearing-types	List Hearing Types (Procedural, Substantive, Decision).	Authenticated
POST	/api/v1/reference/hearing-types	Add a new entry to hearing-types.	Administrator
PATCH	/api/v1/reference/hearing-types/{id}	Amend an entry in hearing-types.	Administrator
DELETE	/api/v1/reference/hearing-types/{id}	Delete an entry from hearing-types.	Administrator

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/reference/status-types	List the 13 case status types (system-defined, read-only).	Authenticated

5.6 Case Registration - Case Entry

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/cases	Create a case entry: appellant, representative, lawyer, case details.	User
GET	/api/v1/cases	Search/list cases (filter by status, type, tax type, member, year; see Section 3.3).	User
GET	/api/v1/cases/{caseId}	Get full case detail.	User
PATCH	/api/v1/cases/{caseId}	Update case entry fields prior to registration.	User
PATCH	/api/v1/cases/{caseId}/appellant	Update Appellant details (Registry No/ID, TIN, addresses, phone, email).	User
PUT	/api/v1/cases/{caseId}/representative	Add/update the person on behalf of the Appellant.	User
PUT	/api/v1/cases/{caseId}/lawyer	Add/update the Appellant's Lawyer details.	User
PATCH	/api/v1/cases/{caseId}/case-details	Update Type of Case, Objection Amount, Reason for Objection, Type of tax.	User
PATCH	/api/v1/cases/{caseId}/mira-decision	Record MIRA's Decision No, Date and attach the decision document.	User
PATCH	/api/v1/cases/{caseId}/appeal	Record Date Appealed and attach the Appellant's Appeal documents.	User
POST	/api/v1/cases/{caseId}/documents	Upload/attach a document to the case.	User
GET	/api/v1/cases/{caseId}/documents	List documents attached to a case.	User
GET	/api/v1/cases/{caseId}/documents/{documentId}	Download an attached document.	User
DELETE	/api/v1/cases/{caseId}/documents/{documentId}	Remove an attached document.	User

5.7 Case Registration - Review, Fee & Registration

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/cases/{caseId}/review/receipt-stamp	Record the case receipt stamp once the case is entered.	User
POST	/api/v1/cases/{caseId}/review/time-limit-check	Record the 60-day legal time-limit check result.	User
POST	/api/v1/cases/{caseId}/review/payment-check	Record the 25% payment/deduction check; if Yes, links the MIRA payment confirmation document.	User
POST	/api/v1/cases/{caseId}/review/document-check	Record the document-completeness check result.	User
POST	/api/v1/cases/{caseId}/review/decision	Record the Tribunal's acceptance decision: register / send back for correction / reject.	Supervisor
POST	/api/v1/cases/{caseId}/review/reject	Reject the case with a reason selected from reference/rejection-reasons.	Supervisor
POST	/api/v1/cases/{caseId}/review/rejection-review	Record the Rejection Reviewed date and decision (Accept/Reject).	Supervisor
POST	/api/v1/cases/{caseId}/registration-fee	Record registration fee payment prior to filing.	User
POST	/api/v1/cases/{caseId}/register	Register the case: auto-generates Case Number (TAT-CA-type/year/seq) and registration date; emails the number to the Appellant.	Supervisor
POST	/api/v1/cases/{caseId}/handover-legal	Hand the registered case file over to the Legal Section.	Supervisor

5.8 Case Assignment

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/cases/{caseId}/assignment	Get all current assignments for a case (Bench, Case Analyst, CRT).	User
PUT	/api/v1/cases/{caseId}/assignment/bench	Assign Bench Chair and two Bench Members from the Members list; records Assigned Date.	Supervisor
PUT	/api/v1/cases/{caseId}/assignment/case-analyst	Assign a Case Analyst (Assessment workflow, Section 5.2 of the Full Documentation).	Supervisor
PUT	/api/v1/cases/{caseId}/assignment/crt	Assign SLO and TO (Legal/CRT workflow, Section 5.3 of the Full Documentation).	Supervisor

5.9 Case Timeline (Status Tracking)

Applies to each of the eleven tracked stages: mira-response, appellant-statement, mira-statement, pre-hearing-meeting, hearings, decision-drafting, review-by-legal, pre-decision-meeting, review-by-panel-members, decision-finalizing, decision-hearing.

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/cases/{caseId}/timeline	Get the full stage timeline (Send/Due/Done dates for all eleven stages).	User
PATCH	/api/v1/cases/{caseId}/timeline/{stage}	Set/update the Send Date and Due Date for a stage.	User
POST	/api/v1/cases/{caseId}/timeline/{stage}/complete	Mark a stage's Date Done; automatically advances the case Status.	User
GET	/api/v1/cases/{caseId}/status	Get the case's current status (one of the 13 status types).	User
GET	/api/v1/cases/{caseId}/status-history	Get the full status change history for a case.	User

5.10 Case Phases

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/cases/{caseId}/phases	List phases selected for a case.	User
POST	/api/v1/cases/{caseId}/phases	Select a phase for the case from reference/phases.	User

5.11 Meetings

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/cases/{caseId}/meetings	Schedule a Case Review Meeting: type, date, time, meeting room.	User
GET	/api/v1/cases/{caseId}/meetings	List meetings scheduled for a case.	User
GET	/api/v1/meetings/{meetingId}	Get a meeting's detail.	User
PATCH	/api/v1/meetings/{meetingId}	Reschedule or update a meeting.	User
POST	/api/v1/meetings/{meetingId}/invitations	Send meeting invitations to selected users.	User
POST	/api/v1/meetings/{meetingId}/complete	Mark the meeting complete with a completion date.	User
DELETE	/api/v1/meetings/{meetingId}	Cancel a meeting.	User

5.12 Hearings

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/cases/{caseId}/hearings	Create a hearing: type (Procedural/Substantive/Decision), date, time. Multiple hearings per case are supported.	User
GET	/api/v1/cases/{caseId}/hearings	List hearings for a case.	User
GET	/api/v1/hearings/{hearingId}	Get a hearing's detail.	User
PATCH	/api/v1/hearings/{hearingId}	Reschedule or update a hearing.	User
POST	/api/v1/hearings/{hearingId}/invitations	Send hearing invitations to selected users.	User
POST	/api/v1/hearings/{hearingId}/complete	Mark the hearing complete with a completion date.	User
DELETE	/api/v1/hearings/{hearingId}	Cancel a hearing.	User
POST	/api/v1/hearings/{hearingId}/transcript	Upload the signed hearing transcript (Legal workflow, Hearing Transcribing stage).	User

5.13 Decision Drafting, Legal Review, Panel Review & Finalizing

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/cases/{caseId}/decision-drafting/start	Record Decision Drafting Started Date.	User
POST	/api/v1/cases/{caseId}/decision-drafting/complete	Record Decision Drafting Complete Date.	User
POST	/api/v1/cases/{caseId}/legal-review	Send the draft for Legal Review: select legal staff, date, attach Word file.	User
GET	/api/v1/cases/{caseId}/legal-review	List the legal review history for a case.	User
POST	/api/v1/cases/{caseId}/legal-review/{reviewId}/complete	Complete a legal review: date, attach reviewed Word file.	User
POST	/api/v1/cases/{caseId}/panel-review	Record Review by Panel Members / assent-dissent outcome.	User
POST	/api/v1/cases/{caseId}/dissenting-opinion	Submit a dissenting Bench Member's draft opinion.	User
POST	/api/v1/cases/{caseId}/decision-finalizing	Record Decision Finalizing date and mark complete.	User

5.14 Dashboard

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/dashboard/summary	Total number of cases, Active cases, Closed cases.	Authenticated
GET	/api/v1/dashboard/cases-by-status	Bar chart data: cases grouped by Status.	Authenticated
GET	/api/v1/dashboard/cases-by-type	Bar chart data: cases grouped by Type of Case.	Authenticated
GET	/api/v1/dashboard/recent-cases	List of recently registered/updated cases.	Authenticated

5.15 Reports & Charts

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/reports/case-register	Full Case Register: all case-entry fields, Case Number, Registered Date.	User
GET	/api/v1/reports/case-list	Case List filtered by list type: by Member, Status, Case Type, Tax Type, or Year.	User
GET	/api/v1/reports/case-stats/by-year	Cases filed, registered and completed, grouped by year.	User
GET	/api/v1/reports/case-stats/by-year-type-status	Number of cases grouped by year, type and status.	User
GET	/api/v1/charts/{chartType}	Bar or Pie chart data for a selected dimension.	User
GET	/api/v1/reports/{reportId}/export	Export a report as PDF, CSV or XLSX.	User

5.16 Notifications

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/notifications	List notifications for the current user.	Authenticated
POST	/api/v1/notifications/{id}/read	Mark a notification as read.	Authenticated
GET	/api/v1/admin/notifications/settings	Get notification trigger configuration (on registration, on status change, 24h before due date).	Administrator
PATCH	/api/v1/admin/notifications/settings	Update notification trigger configuration.	Administrator

5.17 Logs & Audit

Method	Endpoint	Description	Access / Consumer
GET	/api/v1/audit/logins	List login events (who, when, from where).	Administrator
GET	/api/v1/audit/changes	List data-change audit events, filterable by case, user or date range.	Administrator
GET	/api/v1/audit/cases/{caseId}	Full audit trail for a single case.	User

5.18 Files & Documents

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/documents	Upload a document and receive a document ID for attaching to any case entity.	User
GET	/api/v1/documents/{documentId}	Download a document (access-checked against the owning case).	User
DELETE	/api/v1/documents/{documentId}	Delete a document (Administrator or original uploader only).	User

5.19 Outbound Integrations (internal use by the backend)

Method	Endpoint	Description	Access / Consumer
POST	/api/v1/integrations/sms/send	Send an SMS via the SMS gateway for registration confirmation, status change, or due-date reminder.	System / Backend service
POST	/api/v1/integrations/ocr/extract	Submit a scanned appeal document for OCR data extraction/verification.	System / Backend service
GET	/api/v1/integrations/ocr/{jobId}	Poll an OCR extraction job for its result.	System / Backend service

6. Partner API - Other Government Organizations & Apps

Exposed under a separate base path (/partner/v1), behind the API gateway, using the tat-partners realm (Section 4.1). This surface returns only the minimum data required for inter-agency case verification and statement exchange - it is not a mirror of the internal API in Section 5.

Method	Endpoint	Description	Access / Consumer
POST	/partner/v1/oauth/token	Client-credentials token endpoint; exchanges a partner client ID/secret (or mTLS cert) for an access token.	Registered partner
GET	/partner/v1/cases/{caseNumber}/status	Look up a case's current status by its public Case Number - no PII returned.	cases:read
GET	/partner/v1/cases/{caseNumber}/summary	Case summary (type, tax type, status, key dates); Appellant identity fields redacted unless the partner's scope includes them.	cases:read
POST	/partner/v1/cases/{caseNumber}/mira-response	MIRA submits its Response Form directly into the case's Statement Exchange stage.	cases:write:mira
POST	/partner/v1/cases/{caseNumber}/mira-statement	MIRA submits its First Statement directly into the case.	cases:write:mira
GET	/partner/v1/lookups/{listName}	Read-only access to public reference lists (case types, tax types) for a partner's own form validation.	lookups:read
POST	/partner/v1/webhooks/subscriptions	Register a webhook URL to receive case status-change events for cases the partner is party to.	webhooks:manage
GET	/partner/v1/webhooks/subscriptions	List the partner's active webhook subscriptions.	webhooks:manage
DELETE	/partner/v1/webhooks/subscriptions/{id}	Remove a webhook subscription.	webhooks:manage

Every Partner API call is logged (Section 3.7) against the calling partner's client ID, and is subject to the rate limit and scope granted in that partner's onboarding record (Section 4.3). New Partner endpoints are added only when a specific data-sharing agreement requires them - the surface is deliberately kept narrower than the internal API.

7. How the APIs Are Built

7.1 Implementation pattern

Every internal and partner endpoint follows the same layered pattern regardless of which backend option (Section 2.1) is selected, so the catalogue in Sections 5-6 maps 1:1 onto the codebase in any of the three. Only the names of the layers change:

Layer / responsibility	Option A - ASP.NET Core	Option B - Laravel	Option C - Django
Route, HTTP verb, request/response validation, auth policy for Section 4 roles/scopes	Controller + DTO + [Authorize] attribute	Controller + Form Request (validation) + policy/middleware	DRF ViewSet/APIView + Serializer + permission class
Business rule for the endpoint (e.g. 60-day time-limit check, Case Number generation, status auto-advance on Date Done)	Service class (injected via DI)	Service/Action class (invoked from the Controller)	Service function or model method (invoked from the ViewSet)
Only layer that talks to PostgreSQL	Repository over EF Core DbContext	Eloquent Model / Repository	Django ORM Model / QuerySet
Audit logging of every create/update/delete (Section 3.7) before the response is returned	Audit interceptor (middleware)	Model observer / event listener	Model signal (post_save/post_delete) or DRF mixin

7.2 Contract-first documentation

- Whichever option is chosen, the build pipeline generates a single OpenAPI 3.1 document (openapi.json) directly from the code's own annotations - the specification is never hand-maintained separately: Swashbuckle/NSwag for Option A, L5-Swagger/Scribe for Option B, drf-spectacular for Option C.
- The OpenAPI document is published at /api/v1/openapi.json (internal) and /partner/v1/openapi.json (partner), rendered as human-readable docs via Redoc, and used to auto-generate: a typed API client for the frontend (TypeScript for React/Vue; not required for Livewire, which calls the backend in-process), a Postman collection for partner onboarding, and contract tests that fail the build if an endpoint's actual response drifts from its documented schema.
- A partner integrating for the first time is given: the /partner/v1/openapi.json contract, the Postman collection, and this document's Sections 1, 4 and 6 - sufficient to integrate without access to source code, and identical regardless of which backend option produced it.

7.3 Environments & delivery

- Three environments (dev, test/UAT, prod), matching Phase 3 of the implementation timeline, each with its own Postgres instance and Keycloak realm configuration.
- CI pipeline on every pull request: build, unit tests, integration tests against a disposable Postgres (Testcontainers), OpenAPI contract check, then a Docker image is published; deployment to test/prod is a separate, approved release step.
- Frontend and backend are versioned and released independently; the frontend only needs to be re-deployed when it uses a new endpoint, not on every backend change. (Where Livewire, Section 2.2 Option 3, is chosen, this applies to the external REST API's release cadence versus the combined Laravel/Livewire application's own release cadence, since the two are one deployable unit in that case.)

8. Non-Framework Languages & Tools

The core system is built on the backend option selected under Section 2.1 (Option A, B or C - only one) plus the frontend stack in Section 2.2. The items below fall outside that chosen framework and are documented individually, as required.

8.1 OCR microservice language depends on the backend option chosen

OCR integration (Section 3.10 of the Full Documentation / endpoint 5.19 POST /integrations/ocr/extract) is the one component whose "non-framework" status depends on which backend option (Section 2.1) the vendor selects:

If backend option is...	OCR service	Is it a non-framework addition?
Option A (ASP.NET Core) or Option B (Laravel)	Standalone Python 3.12 + FastAPI microservice, called over an internal REST contract	Yes - documented in the table below.
Option C (Django)	Built as a Django app module inside the same backend, using the same Python runtime	No - it is already within the chosen framework; no separate entry is required.

Technology	Where used	Why it is separate from the core framework	Documentation location
Python 3.12 (FastAPI microservice) - only when Option A or B is the chosen backend	OCR integration (Section 3.10 of the Full Documentation / endpoint 5.19 POST /integrations/ocr/extract)	OCR document processing (e.g. Tesseract OCR or a cloud OCR SDK) is most mature in Python's ecosystem; isolating it as its own small service keeps that dependency out of the .NET/PHP backend and lets it be scaled or replaced independently.	A dedicated README.md in the ocr-service repository: setup, environment variables, the internal API contract it exposes to the backend, and its own OpenAPI document.
SQL (DDL/DML)	Database schema, generated primarily via the chosen framework's own migrations (EF Core / Laravel / Django migrations), with any hand-written performance-critical views or indexes	The framework's migration tool generates the day-to-day schema; hand-written SQL is reserved for reporting views (Section 5.15) and is kept intentionally minimal.	Each hand-written SQL file is committed under /database/manual-scripts with a header comment stating its purpose and the migration it is paired with.
PowerShell / Bash	Deployment and environment-setup scripts (Phase 3, CI/CD pipeline steps)	Infrastructure automation is inherently shell-based; kept out of the application codebase.	Each script has an inline header comment (purpose, required parameters, prerequisites) and is referenced from the CI pipeline YAML, which is itself version-controlled.
Dockerfile / YAML (Compose, CI pipeline,	Build and deployment configuration for every service (backend,	Configuration-as-code, not application logic; still needs its own documentation since it defines how	A single deployment.md per repository explains the Dockerfile, required environment variables/secrets,

Kubernetes manifests)	frontend, OCR microservice, gateway)	everything above actually runs.	and how the service is exposed through the API gateway.

Rule for future additions: any tool, library, or language introduced that is not part of the Section 2 stack must be added to this table before it reaches production, with the same four columns completed.

Appendix: Document Control

Version	Date	Description
1.0	13 Sep 2026	Initial API & System Architecture Specification: technology stack, API design standards, authentication/authorization model, internal API catalogue, Partner API surface for other government organizations and apps, build/documentation approach, and non-framework technology register.
1.1	13 Sep 2026	Backend (Section 2.1) restructured into three vendor-choice framework options - ASP.NET Core, Laravel, and Django - each mapped to the same shared components (PostgreSQL, Keycloak, API gateway) and the same API contract; Section 7.1 implementation pattern and Section 8 non-framework register updated to reflect the option selected.
1.2	13 Sep 2026	Frontend (Section 2.2) restructured into three vendor-choice options - React, Vue (both decoupled SPAs, compatible with any backend option) and Livewire (Laravel-coupled, server-rendered - documented as an explicit exception to the frontend/backend separation principle in Section 1.2, only available when Laravel is the chosen backend); Sections 2.3 and 7.2 updated accordingly.

References:

- TAT Case Management System - Full System Documentation & Implementation Timeline (v1.4)
- TAT Case Management System Req.docx (system requirements)
- Registration.docx (Registration workflow and planned SMS/OCR integrations)
- Workflow - Assessment.docx (Case Analyst workflow)
- Workflow - Legal.docx (CRT / Legal workflow)