

IziBuddy Backend API Reference

IziBuddy Backend API Reference

The FastAPI Surface for Frontend Builders

InZeMobile

TABLE DES MATIÈRES

Partie 1 — Foundations

Chapter I — The Backend at a Glance	7
Chapter II — Authentication and Identity	13

Partie 2 — The Template Store

Chapter III — Templates: Store and Lifecycle	21
Chapter IV — Rendering, Validation and Analysis	33
Chapter V — Tags, Presets and Structure	39

Partie 3 — Users and Administration

Chapter VI — Users, Groups and Invitations	53
Chapter VII — Administration	62

Partie 4 — Lifecycle and Quality

Chapter VIII — Versioning, Git and Diff	69
Chapter IX — Scoring and Migration	74

Partie 5 — Platform Services

Chapter X — Real-Time Events, Status and Health	81
Chapter XI — Analytics, Alerts and Diagnostics	85
Chapter XII — Configuration, Tasks and Licensing	91

Appendices

Appendix A — Complete Endpoint Index	99
Appendix B — Data Model Reference	114
Appendix C — Error Handling Reference	121
Appendix D — The Lab: Test and Diagnostic Routes	125
Resources	129

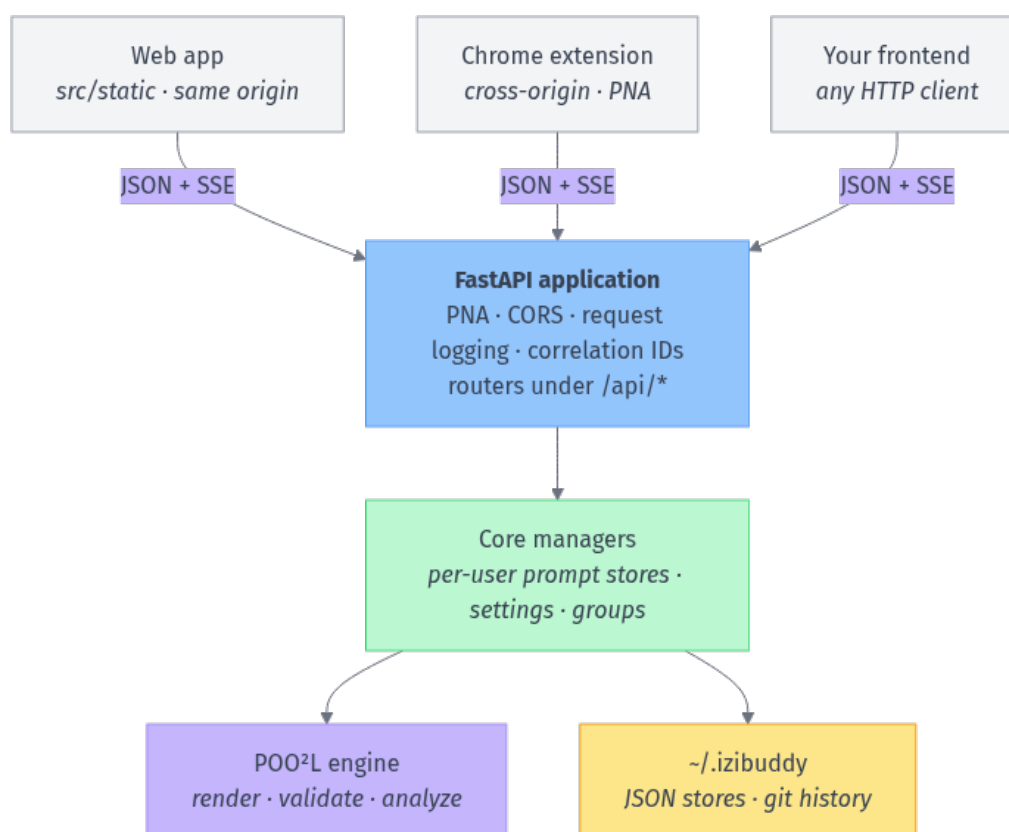
Partie 1

Foundations

Chapter I — The Backend at a Glance

IziBuddy’s backend is a single FastAPI application. Everything the bundled web app and the Chrome extension do — listing templates, rendering them through the POO²L engine, managing users and groups, browsing version history, streaming presence events — goes through the HTTP API documented in this book. There is no private channel : a third-party frontend has access to exactly the same surface as the official clients.

This chapter sets the scene : how the server is laid out, how a client finds and talks to it, and the cross-cutting conventions (CORS, correlation headers, error envelopes, interactive documentation) that every subsequent chapter assumes.



1.1 — Who this book is for

A professional developer who wants to build an independent frontend — a web app, a mobile client, a CLI, an editor plugin — against a running IziBuddy backend. The book assumes fluency with HTTP and JSON, and no knowledge of the IziBuddy codebase. Where an endpoint’s behavior depends on the POO²L language (rendering, validation, includes), the semantics are summarized here and the *POO²L Engine API Guide* and *POO²L Language Reference* hold the full story.

Each endpoint is presented with the same anatomy: method and path, purpose, authentication requirement, parameters, request body, response shape, error statuses, and side effects. Appendix A is the flat index of every route; appendices B and C collect the shared data models and error semantics.

1.2 — Application layout

The application is assembled by `create_app()` in `src/main.py`. It mounts:

- **API routers** — one module per domain under `src/api/`, each contributing a set of routes under `/api/...`. The router map at the end of this chapter ties each prefix to the chapter that documents it.
- **Static frontends** — the bundled web app under `/static` (served from `src/static/`) and the components shared with the Chrome extension under `/common` (served from `src/extension/common/`). `GET /` returns the web app's `index.html`.
- **App-level routes** — `/health`, `/api/info`, `/help` (user help page) and `/dev/diagnostic` (system diagnostic page).

The routers never touch disk or the engine directly: they delegate to *core managers* (`src/core/`) — the templates manager (one store per user), the user settings manager, the group and invitation managers, the analytics tracker, the alert manager. Grammar-level work (rendering, validation, structural analysis) is delegated by those managers to the POO²L engine; the backend deliberately contains no reimplementations of the grammar.

Per-user stores. Most of the API is user-scoped: the acting user is taken from the `X-IziBuddy-User` request header (chapter 2), and prompt data, settings and history are read from that user's store under `~/.izibuddy/users/<username>/`. Two users see two different prompt collections through the same endpoints.

1.3 — Reaching the server

The server binds to `127.0.0.1:8765` by default. Both are configurable at launch (`--host`, `--port`), and the effective port is written once to `.izibuddy/config` (a one-line text file) so that local clients — notably the Chrome extension — can discover it. The file is seeded with the default and not rewritten afterwards, so treat it as a discovery hint rather than a live mirror of a changed port. Unless stated otherwise, every path in this book is relative to the base URL, e.g. `http://127.0.0.1:8765`.

Three launch flags matter to a frontend developer:

Flag	Effect
<code>--lang=fr\ en</code>	Language of backend-generated messages (default <code>en</code>). Error details and human-readable strings in responses follow this setting, not a per-request header.

Flag	Effect
<code>--reload</code>	Development auto-reload. The frontend can detect restarts by polling the <code>boot_id</code> exposed by <code>GET /api/config</code> (chapter 12).
<code>--lab</code>	Mounts the Lab router (<code>/api/tests/*</code> , appendix D). Off by default ; without it those routes plainly 404. There is no way to enable the Lab at runtime.

1.4 — Interactive documentation

A running server self-documents : **Swagger UI** at `/docs` and **ReDoc** at `/redoc` , both generated from the same route declarations this book describes. Use them as the live, always-current complement — this book adds what generation cannot : conventions, flows, event protocols, side effects and the semantics of the values inside the JSON.

1.5 — CORS and Private Network Access

The API is intentionally open at the HTTP layer :

- **CORS** — all origins, methods and headers are allowed, with credentials ; all response headers are exposed. A browser frontend served from any origin can call the API directly.
- **Private Network Access (PNA)** — Chrome blocks requests from public HTTPS pages (for instance `claude.ai` , where the extension runs) to private addresses such as `localhost` unless the server opts in. A dedicated middleware answers PNA preflights (`OPTIONS` with `Access-Control-Request-Private-Network: true` → `204` with `Access-Control-Allow-Private-Network: true`) and stamps `Access-Control-Allow-Private-Network: true` on every response.

Authorization is therefore enforced by the application (chapter 2), never by origin filtering.

1.6 — Request tracing headers

Every request/response pair carries tracing metadata :

- **Correlation ID** — the server reuses the first of `X-Correlation-ID` , `X-Request-ID` or `X-Trace-ID` found on the request, or generates a UUID, and always returns it as `X-Correlation-ID` . Send one from your frontend to correlate client logs with the structured server logs.
- **X-IziBuddy-User** — the identity header (chapter 2). Besides selecting the per-user store, it tags server-side log lines and slow-request alerts with the acting user.

Requests slower than a configurable threshold (default 1000 ms) are logged and can raise a `SLOW_REQUESTS` alert (chapter 11).

Static assets are never cached in development. Responses under `/static/` and `/common/` for `.js`, `.css` and `.html` carry `Cache-Control: no-cache, no-store, must-revalidate` so that a browser refresh always picks up the current frontend code.

1.7 — Response and error conventions

The API speaks JSON exclusively (the only exceptions are the static files, the HTML pages above, and the `text/event-stream` endpoint of chapter 10). Two response idioms coexist :

- **Resource style** — the payload is the resource or list (`GET /api/templates` returns the prompt list). Errors are raised as HTTP statuses with a FastAPI envelope : `{"detail": "..."}` , where `detail` is a string or a structured object.
- **Outcome style** — mutating or long-running operations often return `{"success": true, ...}` with operation-specific fields, and may report domain-level failure with `"success": false` inside a 200 response.

Chapter-level sections always state which idiom an endpoint uses ; appendix C consolidates the error catalogue, including the structured error payloads of the rendering endpoints (engine diagnostics) and validation errors (422 , FastAPI's standard shape).

Backend-generated messages are internationalized server-side : the `--lang` launch flag (persisted in the backend settings) selects French or English for every human-readable string the API emits.

1.8 — Health and identity of the server

Three lightweight endpoints let a frontend confirm what it is talking to :

- `GET /health` — liveness plus component status : which core managers are initialized, and the POO²L engine capability report (`engine.capable: false` signals a stale native engine binary — renders would silently come back empty ; surface this to the user).
- `GET /api/info` — `{"status": "ok", "name": "IziBuddy", "version": "1.0"}` .
- `GET /api/status` — a static liveness probe returning a fixed `{"status": "running", "version": "1.0", "mode": "async"}` (chapter 10); it carries no runtime metrics.

1.9 — Router map

Prefix	Domain	Chapter
<code>/api/auth</code>	Login, sessions, identity	2
<code>/api/templates</code>	Template store, rendering, collections	3, 4

Prefix	Domain	Chapter
/api/tags , /api/tags/v2	Tag taxonomy	5
/api/templates/presets	Parameter presets	5
/api/template-structure , /api/template-builder	Structural editing aids	5
/api/docs	In-app documentation content	5
/api/users	User profiles and settings	6
/api/groups , /api/invitations	Group membership and sharing	6
/api/admin	Administration	7
/api/versions , /api/git , /api/diff	History and comparison	8
/api/scoring	Template quality scoring	9
/api/admin/migration	Store migrations	9
/api/events	Server-sent events, presence	10
/api/status	Server status	10
/api/analytics	Usage analytics	11
/api/admin/alerts	Alert management	11
/api/dev	System diagnostics	11
/api/config	Frontend configuration, live-reload	12
/api/tasks	Background tasks	12
/api/license	Licensing and webhooks	12
/api/tests	Lab (only with <code>--lab</code>)	appendix D

Three historical modules remain in the tree but are *not* mounted (`src/api/projects.py`, `src/api/commands.py`, and the standalone `src/api/http_server.py`); their routes do not exist on a current server and are not documented here.

Skill persistence and explicit publication

Native `document_type` is `prompt` (default) or `skill` ; a conversion preserves UUID and logical ID and creates a version. File, Mongo and SQL adapters preserve it ; the SQL migration is additive and repeatable. SQLite migration is tested ; other SQL dialects require deployment validation against the actual server.

The browser stores drafts, including attachments, in account-scoped IndexedDB. `POST /api/templates/publish` accepts `id`, `token`, `base_revision` and `document`. Compare revision before writing. Conflict returns HTTP 409. The token provides idempotent retry ; pending Git synchronization retains the browser draft. The current process serializes writes ; this is not a multi-process database lock. Only published dependencies enter ordinary rendering. A fresh browser registry provides explicit local preview. Administrative document policy is served by `/api/admin/document-policy` ; render request fields cannot weaken it.

`GET /api/templates/render/{uuid}?target=skill` (or POST with inputs) returns `emission.text`, `emission.name` and relative `emission.files` with base64 bytes. No server path is taken from the user and no script is executed. Browser exports use SKILL.md or a resource ZIP. Publication of source never updates installed output.

Templates and document nature

A **template** is the common reusable entity. Its **Nature** is `prompt`, `skill` or a custom identifier such as `mod` or `acme.review`. Create **New prompt** or **New skill**, then edit Nature in Properties when needed. Changing nature preserves the UUID, references, parameters and history. The leaf template owns its type ; includes never change it. Mixed inheritance can be allowed with warnings or denied.

```
@@@template review_assistant
@@@type skill
@@@description Reusable review instructions
@@@section rules
Separate observations from assumptions.
```

Use `type:acme.review` to search a custom nature. The **Example** flag replaces the old Template flag ; `flag:example` finds these starting examples. `flag:draft` continues to find private drafts. A prompt copies the rendered text, a skill exports SKILL.md and resources, and other types export composed text without claiming a specialized package format.

Chapter II — Authentication and Identity

Everything in the API is scoped to a user, so identity comes first. This chapter describes the two identity mechanisms the backend runs in parallel, the complete login and registration flows a frontend must implement, token mechanics, roles, and the full `/api/auth` route reference.

2.1 — Two parallel identity mechanisms

The backend has **two coexisting ways of knowing who is calling**, used by different routers :

Mechanism	Carrier	Used by	Verified ?
JWT Bearer token	Authorization: Bearer <jwt>	the whole <code>/api/templates/*</code> surface, <code>/api/auth/*</code> protected routes, all of <code>/api/groups/*</code> and <code>/api/invitations/*</code> admin routes, <code>/api/admin/*</code>	Yes — signature and expiry checked
Identity header	X-IziBuddy-User: <username>	<code>/api/users/*</code> settings routes, tags, presets, versions, and several other user-scoped domains	No — the server trusts the header value as-is

The header mechanism predates the JWT layer and remains the identity carrier for a number of data-plane routes : it selects the per-user store but proves nothing. (The templates router itself was migrated to strict JWT — its docstrings still mention the header, but only the token is read.) The backend is designed for trusted local or intranet deployment ; if you expose it further, put an authenticating proxy in front.

Rule of thumb for frontends: send both headers on every request — `Authorization: Bearer <token>` and `X-IziBuddy-User: <username>` (the username exactly as returned by login). That satisfies every router, whatever mechanism it checks.

2.2 — The login flow

A frontend implements sign-in in five steps :

1. **Bootstrap** — `GET /api/auth/status` (anonymous) returns `{"auth_required": true, "has_users": ..., "show_register": ...}`. When `has_users` is false, show the registration form: the **first account ever registered automatically becomes admin**.

2. **Login** — `POST /api/auth/login` with `{"username": ..., "password": ..., "remember_me": ...}`. The `username` field accepts the username **or the email**. Success returns `token` (a JWT) and `user` (the public profile).
3. **Store** the `token` and `user.username` (for instance in `localStorage` — there is no cookie mechanism).
4. **Attach** both headers to every subsequent request, as above.
5. **Probe on boot** — `GET /api/auth/check` with the stored `token` returns `{"authenticated": true|false, "user": ...}` and never errors; use it to decide between the app and the login screen. (`GET /api/auth/me` is the throwing variant: 401 on a bad token.)

Logout is client-side: delete the stored `token`. `POST /api/auth/logout` exists but is a no-op placeholder — there is no server-side session store or token blacklist to clear.

2.3 — Token mechanics

- **Format** — JWT signed with **HS256**; claims `{"sub": <username>, "exp": ..., "iat": ...}`.
- **Signing key** — a random 32-byte secret generated on first boot and persisted in the app config directory (`jwt_secret.key`, mode 600). Tokens survive server restarts; deleting the file invalidates every token.
- **Lifetime** — 24 hours by default; `remember_me: true` extends to **7 days**. (A code comment mentions 30 days; the constant is 7 days.)
- **Renewal** — there is **no refresh endpoint**. When a token expires the next authenticated call returns 401 and the client must re-login. Design your HTTP layer to catch 401s globally and route to the login screen.
- **Passwords** — hashed with `bcrypt`. Five consecutive failed logins lock the account for **15 minutes**; a login attempt during the lock returns the same 401 `"Invalid credentials"` as a wrong password (deliberately indistinguishable). A successful login resets the failure counter.
- **Password changes do not invalidate existing tokens.**

Server-side resolution: the `get_current_user` dependency decodes the `token`, loads the user, and raises 401 `{"detail": "Not authenticated"}` (no credentials), 401 `{"detail": "Invalid or expired token"}` (bad token or deleted user) or 403 `{"detail": "Account is disabled"}` (`is_active` false). `get_admin_user` adds 403 `{"detail": "Admin access required"}` for non-admins. Header-based routes reject a missing `X-IziBuddy-User` header, but not uniformly: routes that read the header manually raise 401 `{"detail": "X-IziBuddy-User header required"}` (admin, alerts) or the shorter 401 `{"detail": "User header required"}` (analytics, versioning), while routes that declare it as a required FastAPI parameter (`Header(...)`, e.g. `/api/events/notify/shutdown`) return a 422 validation error instead (appendix C).

2.4 — Roles

Role	Definition	Grants
Anonymous	no or invalid token	register, login, password reset, invitation validation, registration config, <code>/api/auth/status</code> , <code>/api/auth/check</code> — plus, by omission, several header-based routes
Authenticated user	valid JWT, active account	own profile, password change, group listing and reading, own groups
Group admin	listed in a group's <code>admins</code>	member add/remove, group-admin promote/demote, subgroup attach/detach for that group; parent-group admins act transitively on descendants for template-edit checks
System admin	<code>is_admin: true</code> on the user	all group-admin powers on any group, user CRUD, group CRUD, invitations, server restart, storage configuration

A `is_superuser` flag exists on the user model but no route in the current API requires it.

2.5 — The user object

Login, registration, `/me`, `/check` and the user lists all return the same public profile :

```
{
  "username": "jean.dupont",
  "email": "jean@example.com",
  "is_admin": false,
  "is_superuser": false,
  "is_active": true,
  "groups": ["dev"],
  "status": "active",
  "email_verified": true,
  "has_2fa": false,
  "created_at": "2026-01-13T14:32:00.123456",
  "last_login": "2026-07-12T09:15:00.000001"
}
```

Username and emails are **lowercased at creation**. `status` is one of `active`, `disabled`, `pending`. `has_2fa` is ground-work only — the 2FA routes are not mounted. The `groups` array mirrors the user record; the authoritative source is the groups service (chapter 6), and the admin user list re-resolves groups from it.

2.6 — Route reference — /api/auth

POST /api/auth/register

Open registration (when enabled by deployment policy, and the path taken by the very first user). Anonymous. Body :

Field	Type	Required	Constraints
user-name	string	yes	≥ 3 chars ; letters, digits, <code>-</code> , <code>_</code> ; unique (case-insensitive)
email	string (email)	yes	valid and not already registered
password	string	yes	≥ 6 chars

Response 200 — the account is created **and logged in** :

```
{
  "success": true,
  "message": "User created successfully",
  "user": { ...public user... },
  "token": "<jwt>",
  "is_first_user": true
}
```

`is_first_user: true` also means the account was made admin. Violations of the constraints return 400 with a specific detail string.

POST /api/auth/login

Anonymous. Body: `username` (username **or email**), `password`, `remember_me` (default `false`). Response 200 :

```
{ "success": true, "message": "Login successful", "user": { ... }, "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ5In0:eyJ1b290Ij0iIn0:"
```

Failure is always 401 `{"detail": "Invalid credentials"}` — unknown user, wrong password, locked or inactive account are indistinguishable. Side effects : updates `last_login` and the failure counter (lockout after 5).

GET /api/auth/me

Bearer required. Returns `{"success": true, "user": { ... }}` ; 401/403 per the token rules above.

GET /api/auth/check

Anonymous, optional bearer, **never errors**: {"authenticated": true, "user": { ... }} or {"authenticated": false, "user": null}. Caveat: a valid token for a *disabled* account still reports `authenticated: true` here — the strict check is `/me`.

POST /api/auth/logout

Anonymous no-op; always {"success": true, "message": "Logged out successfully"}. Real logout is deleting the stored token.

POST /api/auth/change-password

Bearer required. Body: `current_password`, `new_password` (≥ 6). Response {"success": true, "message": "Password changed successfully"}; 400 when the current password is wrong or the new one too short. Existing tokens remain valid.

GET /api/auth/status

Anonymous bootstrap: {"auth_required": true, "has_users": true, "show_register": false} — `show_register` is simply «no user exists yet».

GET /api/auth/config/registration

Anonymous; drives the login/register UI:

```
{
  "success": true,
  "self_registration_enabled": true,
  "has_domain_restriction": false,
  "email_enabled": false
}
```

Values come from the server's `izibuddy.ini` (`registration.*`, `email.*`).

User administration (GET /api/auth/users, PUT/DELETE /api/auth/users/{username})

Admin-only (bearer + `is_admin`).

- GET `/api/auth/users` — {"success": true, "users": [...public users...]}, with each user's `groups` re-resolved from the groups service.
- PUT `/api/auth/users/{username}` — partial update; body fields (all optional): `email`, `is_admin`, `is_active`. Response {"success": true, "message": "User updated"}; 400 unknown user. There is no self-demotion guard: an admin *can* remove their own admin flag. Setting `is_active: false` blocks future logins and makes the user's existing tokens fail with 403.
- DELETE `/api/auth/users/{username}` — deletes the account record (400 "Cannot delete your own account" for self-deletion; the user's settings and prompt directories on disk are not touched).

2.7 — Self-registration by email verification

A two-step, anonymous flow, active when self-registration is enabled :

1. `POST /api/auth/register/initiate` with `{"email": ...}` — checks the policy (enabled flag, optional domain whitelist, email not already registered; `400` with a French detail message otherwise), then creates a verification token (default validity **7 days**; previous tokens for the email are invalidated) and emails a link of the form `{base_url}/#/verify-email?token=...`. The base URL honors `X-Forwarded-Proto` / `X-Forwarded-Host`. When SMTP is disabled (the default), the link is only printed to the server console.
2. `GET /api/auth/register/validate/{token}` — pre-fills the completion form: `{"success": true, "valid": true, "email": "user@x.com"}`; `400` for invalid/expired/used tokens. Read-only, does not consume the token.
3. `POST /api/auth/register/complete` with `{"token", "username", "password"}` — creates the account (`email_verified: true, is_admin: false`), consumes the token, and returns the same auto-login payload as `/register`.

2.8 — Password reset

Symmetric two-step, anonymous flow :

1. `POST /api/auth/password-reset/initiate` with `{"email": ...}` — **always returns 200** with a neutral message, never revealing whether the email exists. If it does (and the account is active), a reset token (validity **24 h**) is created and a `{base_url}/#/reset-password?token=...` link is emailed (or console-logged).
2. `GET /api/auth/password-reset/validate/{token}` — same validation shape as above.
3. `POST /api/auth/password-reset/complete` with `{"token", "new_password"}` (≥ 6 chars) — re-hashes the password, **clears any logout**, and consumes the token. Existing JWTs stay valid.

2.9 — What this means for your frontend

- Implement a single HTTP wrapper that injects both identity headers and funnels `401` responses to the login screen.
- Persist `token` + `username` together; they travel as a pair.
- Treat `message` and `detail` strings as display hints only, never as machine-readable signals — some are English, some French, some localized to the server language. Branch on status codes and boolean fields.
- On first-run installations, check `/api/auth/status` and route to registration when `has_users` is false; expect the created account to be the admin.

Partie 2

The Template Store

Chapter III — Templates : Store and Lifecycle

The `/api/templates` router is the heart of the API: around fifty reachable routes covering the prompt CRUD, trash, visibility, search, group sharing, containers (folders, collections, libraries) and import/export. This chapter documents the store and lifecycle; rendering and analysis have their own chapter (chapter 4).

3.1 — Conventions of this router

Authentication is strict JWT Bearer (chapter 2): a missing token is `401 {"detail": "Authentication required"}`, a bad one `401 {"detail": "Invalid or expired token"}`, a disabled account `403 {"detail": "Account is disabled"}`. The username comes from the token's `sub` claim — the `X-IziBuddy-User` header is *not* read here.

Errors are usually HTTP 200. Nearly every handler catches its exceptions and answers `200` with `{"success": false, "error": "..."} (often with an error_type and sometimes a handle for localized UI messages)`. The only non-200 statuses are auth (401/403), Pydantic validation (422), and the group-sharing endpoints, which raise real 400/404 with `{"detail": ...}`. **Always branch on success**, never on the HTTP status alone.

Every user has their own store. The factory resolves the JWT username to a per-user templates manager rooted in that user's directory; there is no cross-user visibility through this API except group-sharing metadata and the admin import path.

Every mutation is a git commit. The default store keeps templates as files in a git repository: each create/update/delete/restore/purge commits and pushes. Bulk endpoints (`/update/bulk`, `/purge-selection`, `/purge-all`) defer to a single commit and push; container deletion and duplication do *not* defer (N commits). Expect mutations to be slower than reads, and git push problems to surface as non-blocking `warnings` (type `git_push_error`) in otherwise successful responses.

No pagination anywhere. `/list`, `/search` and `/scope/list` return the complete arrays.

3.2 — The prompt object

The stored prompt (serialized by `to_dict()`; `name`, `created_at` and `updated_at` duplicate `title`, `created`, `modified` for compatibility):

Field	Type	Meaning
<code>id</code>	string	storage UUID — the stable per-store identity; never the logical id
<code>template_id</code>	string	logical POO²L identifier , canonical UPPER-CASE; unique <i>within its collection path</i> , not globally

Field	Type	Meaning
title / name	string	display name
content	string	full POO ² L source, including the inline declarations (<code>@@@name</code> , <code>@@@param</code> , <code>@@@const</code> , <code>@@@attachment</code> , ...)
tags	string[]	<code>tags[0]</code> is the <i>primary tag</i> — the grouping and sorting key
created / modified	string	ISO timestamps
description , notes	string	metadata
color , short_name	string null	display metadata
locked	bool	locked templates cannot be trashed
important , is_example , wip , no_scoring	bool	flags
disabled	bool	disabled templates refuse to render and block renders that include them
deleted , deleted_at	bool, string null	soft-delete (trash) state
author	string null	author username
shared_with_groups	string[]	group ids the prompt is shared with
order	int	manual sort order within the primary tag (default 999)
references	string[]	ids of referenced (included) templates
parameters	object	formal parameter definitions <code>{name: {type, label, default, required, ...}}</code> — lower-case names; UPPERCASE is reserved for system variables
constants	object	local constants (not propagated to includes)
inherits	string null	parent prompt id (POO ² L object inheritance)
aliases	string[]	alternative identifiers

Field	Type	Meaning
deprecated	string null	non-null = deprecation/migration message
collection	string null	slash-separated container path (Folder/Coll/Sub)
virtual	bool	abstract prompt — instantiating an unimplemented virtual is an engine error
structure_policy	object	{ "diamond": "merge" \ "refuse" \ null } — diamond-include policy
scope, scope_id, owner, visibility	string	multi-tenant fields (personal / group / shared)
source	object	import provenance { "provider", "provider_id" } — foreign ids are never reused
attachments	array	legacy — attachments live inline in content as @@@attachment declarations; this list is kept empty on save

Attachments come in two kinds, both declared inline and mirrored into API-level objects:

- **included** — a file stored with the prompt: {id, type: "included", name?, filename, mime_type, size, data}. data is **always base64 in API responses**; on disk it may be replaced by a content-addressed store reference, which the API re-materializes transparently.
- **to_include** — a file the *end user* must provide at run time: {id, type: "to_include", name?, description, accepted_types: ["pdf"], required}.

3.3 — Reading templates

GET /api/templates/list?includeDeleted=false

The main listing: every live prompt (plus trash when includeDeleted=true) with full metadata, sorted by (primary tag, order, name):

```
{
  "success": true,
  "count": 42,
  "templates": [
    {
      "id": "3f2a...uuid", "template_id": "CODE_REVIEW", "name": "Code review",
      "original_name": "Code review", "description": "...",
      "important": false, "locked": false, "is_example": false,
      "short_name": "CR", "disabled": false, "wip": false, "no_scoring": false,
```

```

    "hidden": false,
    "color": "#ff8800", "tags": ["dev"], "order": 10, "author": "jacques",
    "created_at": "2026-01-02T10:00:00", "updated_at": "2026-06-01T09:30:00",
    "content": "@@@name Code review\n...full P00²L source...",
    "notes": "", "shared": false, "readonly": false,
    "references": [], "deleted": false, "deleted_at": null,
    "parameters": { "lang": { "type": "text", "label": "lang", "default": "fr", "requ
    "constants": {},
    "inherits": null, "collection": "Dev/Reviews"
  }
]
}

```

`hidden` is injected per user (union of the caller's personal hidden set and their groups'); `parameters/constants` are extracted from the inline declarations. The list shape does *not* carry `virtual`, `attachments`, `aliases`, `deprecated` or the multi-tenant fields.

GET /api/templates/get/{template_id}

One prompt in **editing shape**: `content` comes back *clean* (inline declarations stripped) and the declarations arrive as separate fields — `parameters`, `constants`, `attachments` (base64 materialized), plus `name/description` preferring the inline values. The `template_id` path segment is resolved liberally: logical id, storage UUID, display name or alias. Not found is `200 {"success": false, "error": "Template <id> not found"}`. Note: this shape does not include `collection` — read it from `/list`.

Two companions serve the inheritance UI:

- GET /api/templates/get/{template_id}/inheritance-chain — `{"success": true, "chain": [{"id": "BASE", "name": "Base"}, ...]}`, ordered root first.
- GET /api/templates/get/{template_id}/inherited-sections — the sections available from ancestors, for the section-override dropdown.

GET /api/templates/search?q=term

Case-insensitive full-text search over name, content and tags (disabled templates included, trash excluded). `q` is required (422 without it). Returns a reduced shape: `{id, template_id, name, tags, description, important, locked, short_name, disabled, content}`.

Scope listing and groups

GET /api/templates/scope/list?scope= filters by `all`, `shared`, `personal` or `group:{id}` (returns group-shared templates where the store supports them). GET /api/templates/groups returns the caller's groups for the context-selector UI (empty on the default store).

3.4 — Creating and updating

POST /api/templates/create

Body (`name` and `content` required ; everything else optional) :

Field	Type	Default	Notes
<code>name</code>	string	—	display name
<code>content</code>	string	—	POO ² L body ; may already carry inline declarations
<code>description</code> , <code>notes</code> , <code>color</code> , <code>short_name</code>	string	""	
<code>template_id</code>	string	derived from <code>name</code>	sanitized (filesystem-hostile characters removed) and forced UPPERCASE
<code>tags</code>	string[]	[]	<code>tags[0]</code> = primary tag
<code>important</code> , <code>locked</code> , <code>is_example</code> , <code>disabled</code> , <code>wip</code> , <code>no_scoring</code> , <code>virtual</code>	bool	false	flags
<code>attachments</code>	AttachmentRequest[]	[]	see the attachment shapes above
<code>parameters</code>	object	{}	{name: {type, label, default, required, ...}}
<code>constants</code>	object	{}	{NAME: {type, label, value}}
<code>inherits</code>	string null	null	parent id — validated (must exist, no cycle)
<code>collection</code>	string null	null	container path
<code>shared</code>	bool	false	legacy — not supported on the default store

Semantics worth knowing :

- **Uniqueness is per collection** : the create is refused only if another live prompt has the same case-insensitive `template_id` and the same `collection` value.
- **System-variable guard** : a parameter or constant named like a reserved system variable (chapter 4) is refused with `{"success": false, "error": "SYSTEM_VARIABLE_OVERRIDE: ...", "error_type": "system_variable_override"}`.

- **Declaration round-trip**: incoming `content` is stripped of inline declarations, which are re-injected from the explicit fields — sending content that still contains `@@@name` lines is safe, no duplication occurs. The stored content always carries the canonical inline declarations.

Success: `{"success": true, "message": "Template created successfully", "version": "0.1", "warnings": [...]}` — `version` is the initial semantic version; `warnings` appears only when non-empty.

PUT /api/templates/update/{template_id}

Partial update: every field of the create body (plus `new_name` as an alias for `name`) is optional and only non-null fields are applied. Three version-control fields ride along: `major_version: true` bumps to `x+1.0` instead of `x.y+1`, `version_note` attaches a note, `version_operation` marks special operations. Each successful update snapshots a **new version** and returns it: `{"success": true, "message": "Template updated successfully", "version": "1.4", "warnings": [...]}`. Supplying any of `content / parameters / constants / attachments` triggers the same declaration merge-and-reinject cycle as create, with the system-variable guard; attachments are fully replaced and unreferenced stored blobs are garbage-collected.

PUT /api/templates/update/bulk

N per-template patches in one round-trip (mass drag-drop, flag toggles). Body `{"updates": [{"id": "...", "patch": {...}], ...}`. Patch fields are **whitelisted** — `important`, `locked`, `disabled`, `wip`, `no_scoring`, `is_example`, `virtual`, `tags`, `color`, `short_name`, `notes`, `collection`; anything else is silently dropped, and content/parameter changes must use the per-template PUT. There is no cross-template transaction: earlier successes stay committed if a later item fails. Response:

```
{
  "success": true,
  "results": [
    { "id": "uuid-1", "success": true, "warnings": [] },
    { "id": "uuid-2", "success": false, "error": "not_found" }
  ],
  "successCount": 1,
  "errorCount": 1
}
```

`success` is true only when `errorCount` is 0; per-item errors are `empty_or_disallowed_patch`, `not_found`, `update_failed` or an exception string. All changes land in one git commit.

3.5 — Trash lifecycle

Deletion is soft by default; the trash is a first-class state.

Method & path	Effect
DELETE /api/templates/delete/{template_id}	move to trash. Refused for locked templates ({"success": false, "error": "Template is locked"})
GET /api/templates/trash/count	{"success": true, "count": 3}
GET /api/templates/trash/list	trashed templates, list shape with deleted: true and deleted_at
POST /api/templates/restore/{template_id}	restore; optional body {"new_primary_tag": "..."} replaces tags[0] when the original container no longer exists
DELETE /api/templates/purge/{template_id}	permanent deletion of one trashed prompt
DELETE /api/templates/purge-selection	permanent deletion of several — a DELETE with a JSON body {"ids": [...]}; single git commit; response carries success_count / fail_count
DELETE /api/templates/purge-all	empty the trash; single git commit

3.6 — Hidden templates

Independently of the trash, templates can be *hidden* per user or per group (the store behind the eye icon):

- GET /api/templates/hidden — {"success": true, "hidden": ["ID1", "ID2"], "user_hidden": ["ID1"], "group_hidden": {"group-a": ["ID2"]}}.
- POST /api/templates/hidden/{template_id} — toggle. Optional body {"scope": "user"|"group", "group_id": "..."} (group scope requires membership). Returns the state *after* the toggle: {"success": true, "template_id": "...", "is_hidden": true, "scope": "user", "group_id": null}.
- PUT /api/templates/hidden/{template_id} — set explicitly. Body {"hidden": true, "scope": "user", "group_id": "..."}.

Hidden state lives in its own store — changing it does not touch the prompt file or create git commits.

Group resolution. All three handlers resolve the caller's groups from the groups service, which returns a plain list of group-id strings; a group scope is honored only for a group the caller actually belongs to. Note that `/list` already folds the per-user hidden union into each row, so a frontend rarely needs to poll `/hidden` directly.

3.7 — Group sharing

These endpoints are the exception to the 200-only rule: they raise real `400 {"detail": "<reason>"}` on refusal and `404` where noted.

- `POST /api/templates/share-groups/{template_id}` — set the **full list** of groups the prompt is shared with. Body `{"group_ids": ["dev", "qa"]}`. Only the author may modify sharing, and only groups the caller administers may be added. Response `{"success": true, "message": "...", "groups": [ ...re-read list... ]}`.
- `GET /api/templates/sharing/{template_id}` — `{"success": true, "template_id", "author", "groups"}`; `404` if the prompt is unknown.

Several older sharing routes exist but are **non-functional on the default store** and should not be used by new frontends: `POST /share/{id}`, `POST /unshare/{id}` and `POST /duplicate/{id}` always answer `not_supported`; `POST /unshare-group/{id}`, `POST /transfer-ownership/{id}` and `GET /user-templates` fail with HTTP 500 (the underlying manager does not implement the group-sharing surface); a second `GET /sharing/{template_id}` declaration later in the module is dead code. Group visibility in practice flows through `shared_with_groups` + `/scope/list`.

3.8 — Containers: folders, collections, libraries, projects

Containers are **metadata rows keyed by slash path** — the prompt's `collection` field holds the path, and the container row describes the path. `kind` is one of:

- `folder` — organizational only, transparent to POO²L addressing;
- `collection` — qualifies members as `Coll.member` in the engine and may declare OO inheritance (`inherits`);
- `library` — a packaging namespace (see below);
- `project` — transparent to addressing but an include-resolution **scope boundary**.

`GET /api/templates/collections`

```
{
  "success": true,
  "collections": {
    "Dev": { "kind": "folder" },
    "Dev/Reviews": { "kind": "collection", "description": "...", "color": "#123456", "lab
  }
}
```

PUT /api/templates/collections/{path}

Create or update one row (the path may contain `/`). Body fields, all optional with merge-patch semantics: `tags`, `description`, `color` ("" clears), `kind`, `label` (a free display name — the path segment remains the stable address), `capabilities`, `inherits` (collection-to-collection OO parent; "" clears; cycles and kind mismatches rejected), `virtual_templates` (deprecated, ignored for resolution). Nesting rules are validated: collections nest; a library may contain collections; a folder may not live inside a collection or library; a library may not live inside a collection or another library. Violations come back as `200 {"success": false, "error": "<code>"}`.

DELETE /api/templates/collections/{path}?cascade=block|detach

Deletes a container with file-manager semantics: **every member prompt is soft-deleted** (they keep their `collection` field so restoring puts them back), then the metadata of the path and all sub-paths is removed. If other collections inherit from this one, `cascade=block` (default) refuses with `{"success": false, "error": "INHERITS_DESCENDANTS_BLOCK_DELETE", "inherits_descendants": [...]};` `cascade=detach` clears their `inherits` first. Success response:

```
{ "success": true, "success_count": 7, "error_count": 0, "trashed_ids": ["ID1", "..."] }
```

`trashed_ids` enables client-side undo (restore each id, recreate the metadata). Each member soft-delete is its own git commit.

Libraries

A library is a flat packaging manifest *and* an owning namespace path. Names are single identifiers (`^[A-Za-z][A-Za-z0-9_]*$`), canonicalized UPPERCASE. Libraries compose through repeatable, export-only `requires` dependencies (monotonic re-export, not inheritance).

- GET /api/templates/libraries — `{"success": true, "libraries": {"STD-LIB": {"description": "...", "color": null, "requires": ["CORE"], "collections": [], "templates": [], "folder_path": ""}}}`.
- PUT /api/templates/libraries/{name} — merge-patch of the manifest: `description`, `color`, `label`, `requires` (uppercased, deduplicated, each must exist, no self-reference or cycle), legacy `collections` / `templates` member lists, `folder_path` (root or inside a *folder* only). Typed validation errors as `200 {"success": false, "error": "INVALID_LIBRARY_NAME" | "CIRCULAR_LIBRARY_DEPENDENCY" | "UNDEFINED_REFERENCE: ..."} .` Saving also mirrors the library into a `kind: "library"` collection row at `<folder_path>/<NAME>` so it is addressable as a container.
- DELETE /api/templates/libraries/{name} — removes the manifest and its mirrored row; content is untouched (owned templates keep their paths). Refused while another library re-exports it: `{"success": false, "error": "REQUIRES_DESCENDANTS_BLOCK_DELETE", "requires_descendants": [ ... ]}`.

POST /api/templates/container-duplicate

Deep-copies a container subtree under a new top-level name. Body `{"collection": "<path>", "new_name": "<leaf>"} or {"library": "<name>", "new_name": "<leaf>"}` (single segment). Fresh storage UUIDs are minted; copies are unlocked and authored `user`; `inherits` links inside the subtree are re-pathed. Refused if the target exists. Response `{"success": true, "new_path": "Parent/ newName", "count": 12}`. Note: this endpoint commits per copied prompt (no deferred commit).

3.9 — Import and export

All transfers use the **POO²L package** — the normative interchange format of the standard (§7.4), envelope `poo2l_package: "1.0"`. There is no legacy reader or writer: anything that is not a `poo2l_package 1.x` envelope is rejected.

```
{
  "poo2l_package": "1.0",
  "exported_at": "2026-07-12T10:00:00",
  "templates": [
    {
      "storage_id": "3f2a9c1e-uuid",
      "content": "@@@template Dev.CODE-REVIEW\n@@@name Code review\n@@@section task\n...",
      "x_izibuddy": { "tags": ["dev"], "owner": "jacques", "container_path": "Dev" }
    }
  ],
  "collections": [ { "storage_id": "...", "content": "@@@collection Dev\n..." } ],
  "libraries": [],
  "external_dependencies": [ { "kind": 'template', "reference": "Common.DISCLAIMER" } ],
  "x_izibuddy": { "exported_by": "jacques", "include_trash": false,
    "scope": { "users": ["jacques"] }, "folders": {} }
}
```

The record's `content` is the **single source of truth**: no structured field restates what the POO²L text expresses (`title`, `template_id`, `collection`, `inherits`, `virtual`, `parameters`, `constants`, `aliases`, `deprecated` are never separate fields). Application metadata rides the record's `x_izibuddy` block (tags, color, locked, notes, owner, timestamps, ...; defaults omitted); `x_izibuddy.container_path` is a strippable hint of the exact physical location — the content wins on mismatch. Level-0 templates (raw text, no directives) get an identity object (`id`, `name`, `description`); `folder_path` lists the transparent folder segments (omitted at root). Foreign `x_*` blocks are re-emitted **verbatim** — standard §7.4 rights: any tool may read any block, must not modify a foreign block, may delete a whole block only on explicit user request; removing any `x_*` block leaves a conforming package that renders identically. Collection/library objects carry a synthesized `@@@collection / @@@library` block as content and a deterministic uuid5 `storage_id` (importers match by name/path, never by this id). Folders/projects are not packaged entities — they ride the envelope's `x_izibuddy.folders`. Known limitation: attachment payloads (`record: / blob:`) are not packaged; their declarations travel in the content.

GET /api/templates/export?include_trash=&group=&include_my=

Exports the caller's own templates (only those they authored or that are not shared), optionally with a group's templates (`group=<id>`, group admins only — `{"success": false, "error": "Permission denied..."}` otherwise). Response: `{"success": true, "data": <envelope>, "warnings": [...], "total_count": 15}`. The `templates[]` list is flat — no per-user grouping; ownership is carried per record by `x_izibuddy.owner`, and the export scope is traced in the envelope's `x_izibuddy.scope`.

POST /api/templates/import

Imports a POO²L package with a **two-phase protocol**: call with `{"data": ..., "mode": "check"}` to get the conflict report, then `{"data": ..., "mode": "execute", "conflict_resolution": "skip"|"replace"}`. Anything but a `poo2l_package 1.x` envelope is rejected. Conflicts are detected by `template_id` or exact name. Non-admins import everything into their own store with group sharing cleared (no rights escalation via a package); admins route each record to the account named by its `x_izibuddy.owner`.

Check response: `{"success": true, "mode": "check", "new_templates": [...], "conflicts": [...], "new_count": 4, "conflict_count": 1, "warnings": [...]}` — conflict entries carry `conflict_type: "template_id"|"name"` and the existing prompt's name/id. Execute response: `{"success": true, "mode": "execute", "imported": 3, "replaced": 1, "skipped": 0, "containers_created": 2, "containers_skipped": 0, "warnings": [...], "errors": [...], "backup_created": true, "backup_path": "..."}.`

Execute mode creates a **pre-import backup** of the affected stores before any write, then recreates containers (folders, collections, libraries) **before** templates — existing container rows are never overwritten. Imported templates are *localized*: the `storage_id` is dropped (fresh UUID minted) and preserved as `source: {"provider": "import", "provider_id": "<foreign id>"}`.

Single-template and container transfers

- **GET /api/templates/{template_id}/json** — one-template package: the same `poo2l_package` envelope with a single record (no special type marker), wrapped as `{"success": true, "data": <envelope>, "warnings": [...]}`.
- **POST /api/templates/import-single** — accepts a one-template package only, with the same check/execute protocol (has `conflict`, then `action: "added"|"replaced"|"skipped"`).
- **GET /api/templates/container-export?collection=<path> or ?library=<name>** — packages a whole subtree: member templates as `templates[]`, the matched container rows as `collections[]/libraries[]` objects, and `external_dependencies` listing what the subset references without containing. Exactly one of the two parameters must be given.

Foreign `x_*` blocks (extensions)

Foreign `x_*` blocks imported from other tools are stored verbatim on the prompt (`extensions` field) and re-emitted on export. Removing them is an explicit user action, never automatic:

- `GET /api/templates/extensions/namespaces` — namespaces present in the caller's store with carrier counts: `{"success": true, "namespaces": {"x_brol": 3}}`.
- `DELETE /api/templates/{template_id}/extensions/{namespace}` — removes one block from one prompt.
- `POST /api/templates/extensions/purge` — body `{"namespace": "x_brol"}` or `{"all": true}`; purges the namespace(s) across the caller's store and returns `{"success": true, "purged": 3}`.

`POST /api/templates/migrate-ids`

Maintenance endpoint: rewrites templates whose ids contain filesystem-unsafe characters (`? * < > : " / \ |`) — files renamed, indexes and version keys migrated. Response `{"success": true, "scanned": 120, "migrated_count": 2, "migrated": {"OLD?ID": "OLDID"}}`.

3.10 — Practical guidance

- Address templates by **storage UUID** (`id`) in your UI: `template_id` is only unique per collection path, and several templates may share it across collections.
- Treat `GET /get/{id}` + `PUT /update/{id}` as the editing round-trip: the server owns the declaration syntax, your frontend edits clean content plus structured fields.
- After any mutation, refresh from `/list` rather than patching local state: server-side extraction, versioning and git warnings can all adjust what is stored.
- Expect `warnings` arrays on success responses and surface `git_push_error` unobtrusively — the data is saved locally even when the push failed.

Chapter IV — Rendering, Validation and Analysis

Rendering — turning a prompt's POO²L source into the final text with parameters substituted, includes resolved and inheritance applied — is delegated entirely to the **POO²L engine**. The backend contains no reimplementations of the grammar: it builds the engine's inputs (the registry of all the user's templates, the context of parameter values and system variables) and reshapes the engine's outputs into the API contract described here. The engine's own programming interface is the subject of the *POO²L Engine API Guide*; this chapter covers what a frontend sees.

All routes in this chapter live under `/api/templates` and follow that router's conventions (chapter 3): JWT auth, and failures expressed as HTTP 200 with `{"success": false, ...}`.

4.1 — System variables

Before each render, the backend computes the values of the reserved system variables that the engine substitutes for `@@@=NAME` references:

Name	Value
DATE	YYYY-MM-DD (server local time)
TIME	HH:MM:SS
DATETIME	YYYY-MM-DD HH:MM:SS
YEAR , MONTH , DAY	YYYY , 01-12 , 01-31
WEEKDAY	localized day name (French if the user's language is <code>fr</code> , else English)
USER , USERNAME	the JWT username (aliases)
LANG	the user's web-settings language (default <code>fr</code>)
GROUPS	comma-joined group ids of the user (<code>"</code> if none)
GROUPS_ADMIN	comma-joined ids of groups the user administers

These names are **reserved**: declaring a parameter or constant with one of them is refused at save time (`SYSTEM_VARIABLE_OVERRIDE` , chapter 3). The preview endpoint (`/render-text`) computes a reduced set without `USERNAME` , `GROUPS` and `GROUPS_ADMIN`.

4.2 — Warnings, and when a « successful » render fails

The engine returns *warnings* (advisory) alongside the text, and raises *blocking diagnostics* when rendering is impossible. The API reshapes both :

- Each warning is `{"type": "...", "message": "..."} — type` is the lowercase form of the engine's SCREAMING_SNAKE code (`MISSING_DESCRIPTION` → `missing_description`). Warnings may carry `"blocking": true` and a `"handle"` naming the offending include or collection target, which the official UI uses for localized messages.
- The adapter adds a few synthesized types: `invalid_registry_entry` (some *other* prompt in the store failed to parse and was excluded from this render — the render itself still succeeds), and the blocking `undefined_reference` / `render_produced_empty` (guards against silently empty output, typically a stale engine binary — cross-check `GET /health`'s `engine.capable`).

Hard-error promotion. Even when the engine produced text, the API refuses the render (`success: false`, no `rendered` field) if any warning is blocking or its type is one of:

```
unimplemented_virtual_section, override_of_final_section,
override_of_final_template, include_of_virtual_section, ambiguous_include,
system_variable_override, constraint_violation, select_invalid_option
```

The failure payload is :

```
{
  "success": false,
  "error": "<message>",
  "error_type": "<type>",
  "warnings": [ ... ],
  "handle": "<offender, when extractable>",
  "template_id": "...", "name": "..."
}
```

A frontend must never display `rendered` when `success` is false (the field is absent anyway) and should key its error UI on `error_type` + `handle`.

4.3 — GET /api/templates/render/{template_id}

Renders a saved prompt with **system variables only** (no user parameters). The path segment matches `template_id` or storage UUID, case-insensitively. Success :

```
{
  "success": true,
  "rendered": "...final text...",
  "template_id": "CODE_REVIEW",
  "name": "Code review",
```

```

    "tags": ["dev"],
    "is_example": false,
    "warnings": [ { "type": "missing_description", "message": "MISSING_DESCRIPTION: ..."
    "attachments": {
      "included": [ { "id": "a1", "type": "included", "filename": "spec.pdf", "mime_ty
      "to_include": [ { "id": "a2", "type": "to_include", "description": "The contract
    }
  }
}

```

`warnings` appears only when non-empty; `attachments` only when at least one bucket is non-empty — it aggregates the attachments of the prompt **and all its includes**, with `included` data always materialized to base64.

Failure shapes (all HTTP 200), keyed by `error_type` :

Case	Payload
unknown id	<code>{"success": false, "error": "Template with id '...' not found"}</code>
prompt disabled	<code>error_type: "disabled_template" + template_id, template_name</code>
an <i>included</i> prompt disabled	<code>error_type: "disabled_reference" + offender's template_id, template_name</code>
circular include	<code>error_type: "circular_include" + cycle_path: ["A", "B", "A"], template_names</code>
diamond include under <code>refuse</code> policy	<code>error_type: "diamond_include" + template_id, template_name</code>
promoted hard error	the payload shown in the previous section
anything else	<code>{"success": false, "error": "<exception text>"}</code>

4.4 — `POST /api/templates/render/{template_id}`

The instantiation render — same output contract as the GET variant, with a body :

Field	Type	De- fault	Notes
<code>parameters</code>	object of string	<code>{}</code>	user values for the prompt's <code>@@@param</code> s. Merged over the system variables — a user value takes precedence, and can even override <code>DATE</code>
<code>section_states</code>	object of bool	<code>{}</code>	accepted but currently ignored — the handler never forwards it to the renderer, so toggleable sections render with their declared <code>default_on</code> state

4.5 — POST /api/templates/render-text

Preview of raw, possibly unsaved text (the editor's live preview). Body :

Field	Type	De-fault	Notes
text	string	required	raw POO ² L source
template_id	string null	null	when resolvable, that prompt becomes the <i>source prompt</i> : its collection and inherits qualify the text so self / parent and folder-relative includes resolve as if editing that prompt
parameters	object	{}	as above (user wins over system)
section_states	object	{}	ignored, as above

Success is leaner than the saved-template renders: `{"success": true, "rendered": "...", "warnings": [ ... ] | null}` — note that `warnings` is explicitly `null` when empty here, and **attachments are discarded**. Failures use the hard-error payload without `template_id/name` ; the specialized `error_type`s of the saved-template render (circular include, disabled reference...) are not distinguished here — they surface as plain `{"success": false, "error": "..."}`.

4.6 — GET /api/templates/analyze/{template_id}

Static analysis powering the instantiation dialog : which parameters to ask for, which system variables are used, what the prompt includes, its section tree and attachment needs — without rendering. Success :

```
{
  "success": true,
  "analysis": {
    "template_id": "CODE_REVIEW",
    "template_name": "Code review",
    "parameters": {
      "lang": { "type": "select", "label": "Langue", "default": "fr", "required": true },
      "notes": { "type": "textarea", "label": "notes", "default": "", "required": false }
    },
    "system_variables": ["DATE", "USER"],
    "includes": [
      { "template_id": "DISCLAIMER", "template_name": "Disclaimer", "is_disabled": false }
    ],
    "sections": {
      "intro": { "name": "intro", "parameters": ["lang"], "includes": [], "nested_sections": [] }
    }
  }
}
```

```

    },
    "attachments": { "included_count": 1, "to_include_count": 1, "included": [ ... ], "
    "warnings": [ "..."], "errors": [ "..."],
    "has_cycle": false, "cycle_path": []
  }
}

```

Parameter descriptors always carry `type`, `label`, `default`, `required`, plus type-specific extras: `min/max` (number), `options` (select), `rows` (textarea), `minlen/maxlen` (text/textarea), `pattern` (text), and `section` when the parameter only applies to a section. Build your dialog generically from these descriptors — new parameter types are additive.

Unknown prompt: `{"success": false, "error": "...", "error_type": "not_found"}`.

4.7 — Editor support endpoints

GET /api/templates/{template_id}/include-candidates

The reachable `@@@include` targets from a prompt — siblings and everything visible by upward scope walking, bounded by the enclosing *project*. Fetch once when the autocomplete popup opens and filter client-side:

```
{ "success": true, "candidates": [ { "handle": "DISCLAIMER", "kind": 'template' }, {
```

The prompt itself is excluded. When the engine is unavailable: `{"success": false, "error": "engine unavailable", "candidates": []}`.

POST /api/templates/copy-reference

Builds the include snippet for the « copy as reference » action. Body: `{"template_id": "..."} or {"name": "..."} (exact template_id matched first, then exact name):`

```
{ "success": true, "reference": "@@@{include CODE_REVIEW}", "template_name": "Code r
```

4.8 — Implementing an instantiation flow

The canonical sequence for a « use this prompt » feature :

1. `GET /analyze/{id}` — obtain parameter descriptors, `to_include` attachment requirements and structural warnings. Include verdicts come from the engine's own reference check, which does not stop at the first broken target: every section, toggleable ones included, and every candidate of an indirect target are walked, and each failure names the prompt that carries the reference. A missing target of the analyzed prompt is named in `warnings` and flagged `is_missing` ; a failure inside an included prompt keeps the engine's message and flags nothing here; `has_cycle` /

`cycle_path` carry the chain the engine walked. A check that could not run says so (References not checked: ...).

2. Build the form from `parameters` ; pre-fill from a saved preset (chapter 5) when one exists.
3. `POST /render/{id}` with the collected `parameters` .
4. On `success: false` , branch on `error_type` (disabled prompt, missing include, constraint violation...) and use `handle` to name the offender; otherwise deliver `rendered` plus the `attachments.included` files and prompt the user for each `attachments.to_include` requirement.

Chapter V — Tags, Presets and Structure

Templates are organized and edited through five auxiliary services: the **tag taxonomy** (in two generations — a legacy name-keyed API and a first-class v2 entity API), **parameter presets** saved per prompt and per user, the **template-structure catalogue** that drives the creation dialog, the **template-builder configuration** (custom templates and sections, personal and group-shared), and the **documentation content** endpoints that serve the in-app guides. This chapter documents each in turn.

Unless stated otherwise, every endpoint authenticates with the `X-IziBuddy-User` header (chapter 2) and returns `401 {"detail": "X-IziBuddy-User header required"}` when it is missing. Two exceptions are flagged below: the template-structure and docs routes take no identity at all, and the template-builder routes silently fall back to the `default` user.

5.1 — Two generations of tags

The module exposes two routers:

Router	Prefix	Identity	Model
Legacy	<code>/api/tags</code>	tag name (string, may contain <code>/</code>)	entry in the per-user tag index, plus per-template tag strings
v2	<code>/api/tags/v2</code>	tag UUID	first-class <code>Tag</code> entity with ownership, sharing and a per-user favourite flag

They are not independent: legacy metadata mutations (create, rename, delete, color/icon/acronym/short-name/show-icon setters) are **mirrored into the v2 store** as a personal tag of the same name. The modern UI reads v2; the legacy index keeps the per-template tag strings. A consequence worth knowing: the legacy `set_*` endpoints succeed if *either* store accepted the update, so setting metadata on a tag that exists only in v2 (zero templates) still works.

Routing order matters. The legacy router ends with catch-all routes `GET/PUT/DELETE /api/tags/{tag_name:path}` where `:path` matches slashes. The v2 router is registered first so that `/api/tags/v2/...` is not swallowed by the catch-all. The same shadowing applies inside the legacy router: a tag literally named `list`, `create` or `rebuild-index`, or whose name starts with `v2/`, `color/`, `icon/`, `acronym/`, `short_name/` or `show_icon/`, is unreachable through the generic name-keyed routes.

Legacy tag names are hierarchical paths — `dev/python` is one tag whose name contains a slash. URL-encode individual segments; the slash itself is part of the name and of the URL path.

5.2 — Legacy tag routes

Two response models cover the whole legacy surface. `TagResponse` describes a tag :

```
{
  "name": "dev/python",
  "count": 12,
  "color": "#3fb950",
  "icon": "🐍",
  "acronym": "PY",
  "short_name": "Python",
  "show_icon": true
}
```

`count` is the number of the user's templates carrying the tag; `color`, `icon`, `acronym` and `short_name` are display metadata and may be `null`; `show_icon: true` forces the icon onto prompt cards even when the global «tags as icons» setting is off. Mutations return `TagOperationResponse` :

```
{ "success": true, "message": "Tag 'dev' created", "affected_count": 1 }
```

Note that a *domain* refusal (tag already exists, for instance) comes back as HTTP 200 with `success: false`; only malformed input (400), unknown tags (404) and server faults (500) use HTTP error statuses, always with the `{"detail": "..."}` envelope. Several messages and error details are localized to the server language.

GET /api/tags/list

Lists all tags of the current user with usage counts, sorted by count descending, then alphabetically. Response 200 :

```
{ "success": true, "tags": [ { ...TagResponse... }, ... ], "total": 2 }
```

Read-only; 500 on manager failure.

POST /api/tags/create

Creates a new *empty* tag (`count` 0). Body :

Field	Type	Required	Notes
<code>name</code>	string	yes	stripped; empty → 400 {"detail": "Tag name is required"}
<code>color</code>	string null	no	not format-validated here (unlike the color setter)

Field	Type	Required	Notes
icon	string null	no	emoji

Returns `TagOperationResponse` with `affected_count: 1` ; if the tag already exists, 200 with `success: false` and an explanatory message. Side effects : writes the legacy index and dual-writes a personal v2 tag.

POST /api/tags/rebuild-index

Rebuilds the legacy tag index by rescanning all prompt files — the recovery endpoint for a corrupted or out-of-sync index. No parameters. Response is a `TagOperationResponse` whose `affected_count` is the number of templates indexed, e.g. "Index rebuilt: 148 templates, 23 tags".

PUT /api/tags/color/{tag_name}

Sets or clears the tag color. Body `{"color": "#3fb950"}` ; `null` clears. A non-null value must start with `#` and be 4 or 7 characters (`#RGB` / `#RRGGBB`), otherwise 400 {« detail » : « Color must be in hex format (#RGB or

RRGGBB) »}. 404` if the tag exists in neither store. Mirrors into v2.

PUT /api/tags/icon/{tag_name}

Sets or clears the emoji icon. Body `{"icon": "🐸"}` ; `null` or empty clears. 404 / 500 as above. Mirrors into v2.

PUT /api/tags/acronym/{tag_name}

Sets or clears the card acronym. Body `{"acronym": "py"}` — the value is **uppercased server-side** and must be at most 4 characters after stripping (400 {"detail": "Acronym must be 4 characters or less"}). Mirrors into v2.

PUT /api/tags/short_name/{tag_name}

Sets or clears the short display name (used where the full name would overflow). Maximum 15 characters after stripping (400). Mirrors into v2.

PUT /api/tags/show_icon/{tag_name}

Sets the per-tag `show_icon` flag. Body `{"show_icon": true}` (default `false`). Mirrors into v2.

GET /api/tags/{tag_name}

Returns the single `TagResponse` for a tag name, `404` if unknown. This is the catch-all route that shadows anything not matched above.

PUT /api/tags/{tag_name}

Renames a tag across **all templates** that use it. Body `{"new_name": "development"}` (non-empty after strip, else `400`). If `new_name` already exists the operation is a **merge**: both populations end up under the new name. `affected_count` is the number of prompt files rewritten. Side effects: rewrites the tag arrays in every affected prompt, invalidates the prompt cache, and mirrors the rename onto the v2 personal tag (even when zero templates were touched).

DELETE /api/tags/{tag_name}

Deletes an **unused** tag. A tag with `count > 0` is refused with `400` and a localized detail — remove it from all templates first. `404` if unknown. Mirrors the deletion onto the v2 personal tag of the same name.

5.3 — v2 tag routes

A v2 tag is a stored entity identified by UUID, owned by exactly one of a user (*personal tag*) or a group (*group tag*). Canonical shape:

```
{
  "id": "8f14e45f-ceea-467f-a8f0-1c6b1f0a2d31",
  "name": "python",
  "owner_user": "jean.dupont",
  "owner_group": null,
  "description": "Templates liés à Python",
  "color": "#3fb950",
  "icon": "🐍",
  "acronym": "PY",
  "short_name": "Python",
  "show_icon": true,
  "synonyms": ["py", "python3"],
  "created_at": "2026-07-12T09:15:00.123456",
  "created_by": "jean.dupont"
}
```

Visibility is the union of the caller's personal tags and the tags owned by any of the caller's *effective* groups (transitive membership). Authorization is role-based: mutating a personal tag requires being its owner; mutating a group tag requires being an admin of the owning group. Error details are localized strings; the envelope is always `{"detail": "..."} .`

GET /api/tags/v2/visible

All tags visible to the caller, as a bare JSON **array** (no envelope), each hydrated with the caller's favourite flag:

```
[ { ...tag..., "important_for_me": true }, ... ]
```

POST /api/tags/v2

Creates a tag. Body fields (all optional except `name`): `name` (stripped), `color`, `icon`, `description`, `acronym`, `short_name`, `show_icon` (default `false`), `synonyms` (entries stripped, blanks dropped), `owner_group`. With `owner_group` set the tag is group-owned and the caller must be an admin of that group (403 otherwise); without it the tag is personal. Unlike the legacy API, v2 applies **no** hex-format or length validation to `color`, `acronym` or `short_name`. Returns the created tag dict — the server assigns `id`, `created_at` and `created_by`. 400 if the save failed or the name was empty.

PUT /api/tags/v2/{tag_id}

Partial metadata update with **clear-versus-omit semantics**: a field omitted from the JSON is left untouched; a field explicitly sent as `null` is *cleared* (e.g. `{"icon": null}` removes the icon). Two exceptions: `name` can never be cleared (a null name is ignored) and `show_icon: null` is ignored. Accepts the same fields as creation minus `owner_group`. Returns the full updated tag. 404 unknown, 403 not owner / not group admin, 500 save failure.

DELETE /api/tags/v2/{tag_id}

Deletes a tag with the same ownership rules. The deletion **cascades**: prompt links and per-user favourite flags for the tag are removed. Response `{"success": true}`.

POST /api/tags/v2/{tag_id}/important

Sets the caller's per-user favourite flag. Body `{"flagged": true}`. No ownership needed, but the tag must be *visible* to the caller (403 otherwise). Response `{"success": true, "flagged": true}`.

POST /api/tags/v2/{tag_id}/share/{group_id}

Shares a **personal** tag with a group. The original personal tag is kept; a new group-owned **copy** is created (new UUID, same name and metadata, synonyms not copied). Idempotent: if the group already owns a tag with that name (case-insensitive), the existing group tag is returned with the message "Group already owns a tag with this name". The caller must be the tag's owner and an admin of the target group (403 for either failure). Response:

```
{ "success": true, "tag": { ...group-owned copy... }, "message": "Shared with group" }
```

POST /api/tags/v2/{source_id}/merge-into/{target_id}

Merges two tags : every prompt linked to the source is re-linked to the target, the caller's favourite flag is transferred if set on the source, and the source is deleted. Merges are **same-scope only** — both personal and owned by the caller, or both owned by the same group with the caller as (transitive) admin ; cross-scope merges are refused. All failures come back as 400 with an explanatory detail (including «not found» cases — this endpoint never returns 404). Response :

```
{ "success": true, "moved": 7, "message": "Merged" }
```

GET /api/tags/v2/prompt/{template_id}

The v2 tag ids attached to a prompt: `{"tag_ids": ["8f14e45f-...", ...]}`. An unknown prompt yields an empty list, not an error.

POST /api/tags/v2/prompt/{template_id}/assign

Assigns a tag to a prompt **by name**, with *smart assignment* : the prompt is always linked to a **personal** tag owned by the caller. If the caller does not own a tag of that name, one is created on the fly, snapshotting the display metadata from any visible group tag of the same name — so if the group later deletes its tag, the user keeps a working copy. Body `{"name": "python"}`. Returns the (possibly newly created) personal tag dict. 400 on empty name or save failure.

DELETE /api/tags/v2/prompt/{template_id}/tag/{tag_id}

Unlinks a tag from a prompt. Response `{"success": true}` ; 500 if the removal fails.

5.4 — Parameter presets

Router prefix: `/api/templates/presets`. A preset is a **named bundle of parameter values** for one prompt, saved per user, so that a frequently used combination can be re-applied in one click. The server stores one JSON file per (user, prompt) pair and treats the payload as **opaque** — the structure below is a frontend convention, not a server contract :

```
{
  "quick": { "lang": "fr", "tone": "formal" },
  "verbose": { "lang": "fr", "tone": "friendly" },
  "_default": "quick"
}
```

`_default` names the preset auto-applied when the prompt's parameter dialog opens. Because prompt ids are sanitized into file names (filesystem-hostile characters replaced, 200-character cap), two ids that sanitize identically would share a preset file — an edge case to keep in mind when generating ids.

GET /api/templates/presets/{template_id}

Returns `{"success": true, "presets": { ... }}`. A missing *or corrupt* preset file yields `200` with an empty `presets` object — corruption is silently absorbed, never surfaced as an error.

PUT /api/templates/presets/{template_id}

Whole-document replace. Body `{"presets": { ... }}` — send the complete object, including `_default`. Response `{"success": true, "message": "Presets saved"}`.

DELETE /api/templates/presets/{template_id}

Deletes all presets of the prompt. Idempotent. Response `{"success": true, "message": "Presets deleted"}`.

GET /api/templates/presets/

Lists the prompt ids (file stems, i.e. *sanitized* ids, sorted) that have saved presets for the caller: `{"success": true, "template_ids": [ ... ]}`. Note the trailing slash — without it the path would be read as a `template_id`.

5.5 — The template-structure catalogue

Router prefix: `/api/template-structure` — **no authentication**. This is the server-wide, read-mostly catalogue behind the «new prompt» dialog: the available *sections* (role, context, task, ...) and the *standards* (named section bundles), loaded from a YAML file on the server with an mtime cache and a built-in fallback.

All read routes accept a `lang` query parameter (`fr` default, `en` supported; anything else behaves like `fr`). Localization happens server-side: with `lang=en` the English variants are promoted into the base keys and the `*_en` twins are stripped from the response. One caveat: a section's `singular` label is never promoted for English — English clients receive the French `singular`.

A section object (fields vary by section):

```
{
  "label": "Tâches",
  "singular": "Tâche",
  "type": "list",
  "item_type": "textarea",
  "allow_reference": true,
  "unique": true,
  "allow_duplicates": false,
  "placeholder": "Ex: Analyse ce code..."
}
```

`type` is `"textarea"` or `"list"` ; `unique: true` means at most one instance of the section per prompt; `allow_reference` means the section content may reference other templates. A standard object:

```
{ "label": "Template standard", "description": "Structure universelle", "sections": ["r
```

The four routes :

Method & path	Returns
GET /api/template-structure/config?lang=	<code>{"success": true, "config": {"version", "default", "sections": {...}, "standards": {...}}</code> — the whole catalogue
GET /api/template-structure/sections?lang=	<code>{"success": true, "sections": {...}}</code>
GET /api/template-structure/standards?lang=	<code>{"success": true, "default": "standard", "standards": {...}}</code>
POST /api/template-structure/reload	Invalidates the YAML cache and re-reads the file ; <code>{"success": true, "message": "..."} </code>

All error cases are `500 {"detail": "..."}` .

5.6 — Template-builder configuration

Router prefix: `/api/template-builder` . Where the template-structure catalogue is server-wide, the builder configuration is **per user**: custom templates and custom sections for the TemplateBuilder UI, stored as YAML in the user's directory, plus **group-shared templates**.

Auth caveat. *Unlike every other service in this chapter, these routes do not reject a missing `X-IziBuddy-User` header: they silently fall back to the user `default` . An unauthenticated client therefore reads and writes the `default` user's builder configuration rather than receiving a 401. Always send the header.*

Two validated models are used by the single-item routes :

`TemplateData` — `label` (string, required), `sections` (ordered list of section keys, required), `customSections` (optional object holding inline definitions of non-standard sections the template uses).

`SectionData` — `label` (string, required), `type` (string, default `"textarea"`, by convention `"textarea"` or `"list"` — not enum-enforced), `singular` (optional item label for `list` sections).

The bulk `POST` routes accept raw JSON objects without per-item validation; the config route accepts `{"templates": ..., "sections": ...}` where an omitted half leaves the corresponding file untouched.

Personal configuration

Method & path	Effect
<code>GET /api/template-builder/config</code>	Full config: <code>{"success": true, "username", "templates": {...}, "sections": {...}}</code> (empty objects when nothing saved)
<code>POST /api/template-builder/config</code>	Save whole config (only provided halves written); <code>{"success": true, "message": "Configuration saved"}</code>
<code>GET /api/template-builder/templates</code>	<code>{"success": true, "templates": {...}}</code>
<code>POST /api/template-builder/templates</code>	Replace all templates (raw object body)
<code>PUT /api/template-builder/templates/{key}</code>	Add/update one template (body <code>TemplateData</code>)
<code>DELETE /api/template-builder/templates/{key}</code>	Delete one template — idempotent
<code>GET /api/template-builder/sections</code>	<code>{"success": true, "sections": {...}}</code>
<code>POST /api/template-builder/sections</code>	Replace all sections (raw object body)
<code>PUT /api/template-builder/sections/{key}</code>	Add/update one section (body <code>SectionData</code>)
<code>DELETE /api/template-builder/sections/{key}</code>	Delete one section — idempotent

Failures are `500` with a specific detail string (`"Failed to save configuration"`, `"Failed to save template"`, ...).

Group-shared templates

When a template is shared into a group the server stamps it with `sharedBy` (username) and `sharedAt` (ISO timestamp).

`GET /api/template-builder/group-templates` — the templates of *all* the caller's groups, keyed by group id, with display names resolved:

```
{
  "success": true,
  "groupTemplates": {
    "team-dev": {
      "name": "Équipe Dev",
      "templates": {
        "code-review": {
          "label": "Code review",
          "sections": ["context", "task", "rules"],
          "sharedBy": "alice",
          "sharedAt": "2026-07-01T10:30:00.000000"
        }
      }
    }
  }
}
```

GET /api/template-builder/group/{group_id}/templates — one group’s templates; membership required (403 otherwise).

POST /api/template-builder/group/{group_id}/templates/{key} — share (add or overwrite) a template into a group; body `TemplateData`. Restricted to group admins and system admins (403); 404 if the group does not exist. The success message quotes the group’s display name.

DELETE /api/template-builder/group/{group_id}/templates/{key} — remove a shared template; same authorization; idempotent at the storage layer.

5.7 — Documentation content

Router prefix: /api/docs — **no authentication**. These routes serve the in-app user guides as **raw Markdown**, not JSON: a success response has `Content-Type: text/markdown; charset=utf-8` and the body is the Markdown source, which the frontend renders client-side. Error responses fall back to the JSON envelope: 404 {"detail": "Documentation not found"}, 500 {"detail": "Error loading documentation"}.

The routes are a **fixed whitelist** — there is no catch-all page parameter, so path traversal is impossible, and a new page requires a new route :

Path	Content
GET /api/docs/guide/getting-started	Getting-started guide
GET /api/docs/guide/quickstart	5-minute quick start
GET /api/docs/guide/templates	Template management guide
GET /api/docs/guide/syntax	POO ² L syntax guide
GET /api/docs/guide/categories	Categories guide

Path	Content
GET /api/docs/guide/views	Views and filters guide
GET /api/docs/guide/import-export	Import/export guide
GET /api/docs/guide/shortcuts	Keyboard shortcuts
GET /api/docs/guide/saisie-assistee	Typing assistance
GET /api/docs/guide/extension	Chrome extension guide
GET /api/docs/guide/customization	UI customization guide
GET /api/docs/guide/tips	Contextual tips listing
GET /api/docs/guide	Legacy alias for getting-started
GET /api/docs/cgu	Terms of service
GET /api/docs/privacy	Privacy policy
GET /api/docs/changelog	Changelog
GET /api/docs/backend	Backend documentation

5.8 — Summary of conventions

Concern	Tags legacy	Tags v2	Presets	Structure	Builder	Docs
Auth	header, 401	header, 401	header, 401	none	header, falls back to default	none
Resource identity	tag name (path, / allowed)	tag UUID	prompt id (single segment)	YAML keys	template/ section key	fixed routes
Envelope	success / message models	bare dicts and arrays	{success, ...}	{success, ...}	{success, ...}	raw Mark-down
Error payload	{"detail": str}	{"detail": str} (localized)	{"detail": str}	{"detail": str}	{"detail": str}	{"detail": str}

Partie 3

Users and Administration

Chapter VI — Users, Groups and Invitations

Beyond the account itself (chapter 2), three services shape the collaborative side of IziBuddy: **per-user settings** (`/api/users`), **groups** (`/api/groups`) and **invitations** (`/api/invitations`). Settings routes use the `X-IziBuddy-User` header; groups and invitations are JWT-authenticated.

6.1 — Per-user settings — `/api/users`

Each user has two settings documents, stored on disk under `~/.izibuddy/users/{username}/settings/`: `web.json` (the web app) and `extension.json` (the Chrome extension). A third, global document (`backend.json`) holds backend-wide settings. Usernames are sanitized for path safety (alphanumerics, `-`, `_`, `.`; an empty result becomes `default`).

Trust model. These routes perform no token verification: `/me` routes read `X-IziBuddy-User`, and the `per-username` routes trust the path outright — any client can read or write any user's settings, including the global backend settings. This is consistent with the backend's trusted-deployment posture (chapter 2), but a frontend should still treat usernames as sensitive input and always operate through `/me`.

The settings update model

Every `PUT` in this router shares one partial-update body: fields omitted or sent as `null` are ignored, every non-null field is merged into the stored document, and the server stamps `lastAccess`. A consequence worth noting: a stored key can never be reset to `null` through the API — but `false` and empty values work fine.

Field	Type	Meaning
<code>lang</code>	string	UI language (<code>"fr"</code> , <code>"en"</code>)
<code>theme</code>	string	legacy <code>"light"</code> / <code>"dark"</code> (superseded by <code>active_theme</code>)
<code>ui_level</code>	string	legacy 3-level UI — kept for backward compatibility only
<code>active_preset</code>	string	feature preset: <code>"simple"</code> , <code>"standard"</code> , <code>"advanced"</code> , <code>"expert"</code>
<code>features</code>	object of bool	per-feature toggle overrides (the 49-feature registry); empty = preset-driven

Field	Type	Meaning
feature_configs	object	named UI configs: {name: {features: {...}, base_preset: "standard"}}
active_config	string	name of the active named config (null = preset-driven)
selection_mode	string	"simple" (click opens) or "advanced" (click selects)
enable_scoring	bool	show or hide the scoring flag and filter
default_no_scoring	bool	default no_scoring for new templates
container_colors	object	{folder\collection: {light\dark: "#hex"}} tile backgrounds
active_theme	string	name of the active color theme
themes	object	{name: {base: "light"\dark", builtin: bool, colors: {"--izi-token": "#hex"}}
table_config	object	{hidden: [ids], widths: {id: px}, order: [ids]}
pool_max_level	int	personal POO ² L editing-complexity ceiling, 0-5

GET responses merge the stored file with server defaults, so a frontend always receives a complete document (e.g. lang: "en", theme: "light", active_preset: null, features: {}, default container colors, ...).

Routes

Method & path	Auth	Effect
GET /api/users	none	{"users": ["alice", ...], "count": N} — usernames having a settings directory
GET /api/users/me	header	{"username", "web": {...}, "extension": {...}} — both settings bundles
PUT /api/users/me/web	header	partial update of web.json ; returns {"success": true, "username", "settings": {...full merged...}} — this is how the frontend persists active_preset, features, themes

Method & path	Auth	Effect
PUT /api/users/me/extension	header	same for extension.json
GET /api/users/me/is-admin	header	{"username": "alice", "is_admin": true}
GET /api/users/{username}	none	{"username", "exists", "web_settings", "extension_settings", "templates_path"} ; never 404s (exists: false with defaults); creates the user's templates/ directory as a side effect
POST /api/users/{username}	none	provision the settings directory with defaults (idempotent; created: true false). Settings-level only — does not create an auth account
GET /api/users/{username}/web	none	the merged web settings object, bare (no wrapper)
PUT /api/users/{username}/web	none	update any user's web settings
GET /api/users/{username}/extension	none	bare extension settings
PUT /api/users/{username}/extension	none	update any user's extension settings
GET /api/users/settings/backend	none	global backend settings ({"lang": "en", "admins": []} ; admins is legacy — authoritative admin state is is_admin on the account)
PUT /api/users/settings/backend	none	update global backend settings

Method & path	Auth	Effect
POST /api/users/admins/{username}	header, admin	promote to admin (<code>is_admin: true</code>). Bootstrap exception: when no admin exists yet , the first caller can self-appoint. Returns the updated admin list
DELETE /api/users/admins/{username}	header, admin	demote (<code>is_admin: false</code>); no self-demotion guard

Shadowed route. A `GET /api/users/admins` listing exists in the code but is declared after `GET /api/users/{username}` — the literal path `admins` matches the parameterized route first, so the listing is unreachable: you get the user-info payload for a pseudo-user named `admins`. Use `GET /api/auth/users` (chapter 2) to enumerate admins.

Web settings writes are durable by design: the server writes with `fsync` and verifies by reading back before answering.

6.2 — Groups — /api/groups

Groups drive sharing: group tags (chapter 5), group templates (chapter 5), group-shared templates (chapter 3). **All routes require a valid JWT**; mutations additionally require system admin or, where noted, *group admin* (listed in the group's `admins` array). The group-scoped guard returns `404 "Group not found"` for unknown groups and `403 {"detail": "You must be a System Admin or Group Admin to perform this action"}` otherwise.

The public group shape:

```
{
  "id": "dev",
  "name": "Développeurs",
  "description": "",
  "admins": ["alice"],
  "members": ["bob", "carol"],
  "subgroups": ["dev-front"],
  "created_at": "2026-01-10T08:00:00",
  "created_by": "alice"
}
```

Group ids are lowercased at creation. Nested groups form a **strict tree**: a group has at most one parent and cycles are rejected. *Effective* membership is transitive, LDAP-style — members of subgroups count as members of every ancestor; a parent group's admin is transitively treated as admin of descendants for template-edit checks.

Reading

Method & path	Auth	Returns
GET /api/groups	user	<code>{"success": true, "groups": [{..., "member_count"}], "count"}</code> — all groups; <code>member_count</code> counts direct members only
GET /api/groups/me	user	groups the caller belongs to (member or admin)
GET /api/groups/me/admin	user	groups the caller <i>administers</i> (compact shape <code>{id, name, description, member_count}</code> where <code>member_count</code> is the members $\cup$ admins union) — the list to offer in « share with... » UIs
GET /api/groups/{group_id}	user	one group with <code>member_count</code> ; 404 unknown
GET /api/groups/{group_id}/members?scope=	user	detailed member list; <code>scope</code> is <code>direct</code> (default) or <code>effective</code> (transitive). Members are sorted group-admins-first then by username; each entry: <code>{username, email, is_active, is_group_admin, status}</code> . Members without an auth account are silently skipped. 400 for other scope values
GET /api/groups/{group_id}/subgroups	user	<code>{"success": true, "group_id", "subgroups": ["dev", "qa"]}</code> — direct children
GET /api/groups/{group_id}/tree	user	full recursive subtree: <code>{"tree": {id, name, description, subgroups: [...]}}</code>
GET /api/groups/user/{username}	user (self) or admin	a user's groups; 403 "Cannot view other user's groups"

Mutating

Method & path	Auth	Effect
POST /api/groups	system admin	create. Body: <code>id</code> (≥ 2 chars, letters/digits/ - / _ , unique), <code>name</code> , <code>description</code> (optional). The creator becomes the first group admin. 400 on constraint violations

Method & path	Auth	Effect
PUT /api/groups/{group_id}	system admin	partial update ; body fields name , description , admins (full replacement), members (full replacement)
DELETE /api/groups/{group_id}	system admin	delete ; refused (400) while members is non-empty — admins-only groups can be deleted
POST /api/groups/{group_id}/members/{username}	group/system admin	add a member. The username is <i>not</i> validated against the account store
DELETE /api/groups/{group_id}/members/{username}	group/system admin	remove from members and admins. A group admin cannot remove themselves (400 — ask a system admin) ; a system admin can
POST /api/groups/{group_id}/admins/{username}	group/system admin	promote an existing member (400 "User is not a member of this group"); idempotent
DELETE /api/groups/{group_id}/admins/{username}	group/system admin	demote (stays a member) ; group admins cannot self-demote ; idempotent
POST /api/groups/{parent}/subgroups/{child}	admin of parent	attach a child. 400 if either group is missing, the child already has a parent, or a cycle would form
DELETE /api/groups/{parent}/subgroups/{child}	admin of parent	detach a direct subgroup
PUT /api/groups/user/{username}	system admin	replace a user's full membership set. Body {"groups": ["dev", "qa"]} ; the user is first removed from every group (including admin slots), then re-added as plain member ; unknown ids silently skipped. Returns the resulting id list

Dead route. GET /api/groups/check-access appears in the source but is shadowed by GET /api/groups/{group_id} (you get 404 "Group not found"), and its backing methods don't exist. Rely on the template-level permission fields (chapter 3) instead.

6.3 — Invitations — /api/invitations

Invitations let an admin onboard users by email with pre-assigned groups. They are persisted server-side with URL-safe random tokens; states are `pending`, `accepted`, `revoked`, and computed `expired`. The public shape **truncates the token** — the full token is returned only once, at creation :

```
{
  "token": "aB3xYz9q...",
  "email": "new@company.com",
  "groups": ["dev"],
  "invited_by": "alice",
  "created_at": "2026-07-01T10:00:00",
  "expires_at": "2026-07-08T10:00:00",
  "status": "pending",
  "accepted_at": null,
  "is_valid": true
}
```

Admin routes (JWT + admin)

Method & path	Effect
GET /api/invitations	all invitations, newest first (every state)
GET /api/invitations/pending	only currently valid ones
POST /api/invitations	create. Body: <code>email</code> (required ; must have no pending invitation and no existing account), <code>groups</code> (optional id list), <code>expires_days</code> (default 7). Response includes the full token — the only time it is revealed. No email is sent by this endpoint : the admin distributes the link (<code>{base_url}/#/accept-invitation?token=...</code>)
DELETE /api/invitations/{token}	revoke a <i>pending</i> invitation (<code>400</code> for other states)
GET /api/invitations/by-email/{email}	the pending valid invitation for an email ; <code>404</code> if none
POST /api/invitations/cleanup	purge accepted/revoked/expired invitations older than 30 days ; returns <code>removed_count</code>

Method & path	Effect
POST /api/invitations/batch-import	bulk-create from CSV (below)

Anonymous routes

GET /api/invitations/validate/{token} — validates without consuming: {"success": true, "valid": true, "email", "groups", "expires_at"} ; 400 with a specific message for invalid, revoked, used or expired tokens.

POST /api/invitations/register — creates the account from an invitation. Body: token, username (standard rules), password (≥ 6). The email comes **from the invitation**, never from the client. On success: public user + auto-login JWT + the groups joined ; the invitation is marked accepted.

Known defect (as of this writing). After creating the account, the handler calls a persistence method that no longer exists, producing an HTTP 500 after account creation but before the invitation is consumed and groups are assigned. Treat a 500 from this endpoint as «the account probably exists — try logging in», and expect group assignment to be incomplete.

Batch import

POST /api/invitations/batch-import accepts multipart/form-data :

Field	Type	De-fault	Notes
file	CSV upload	required	.csv only, max 1 MB
send_emails	bool	false	when true, each created invitation is emailed with its acceptance link
ex-pires_days	int	7	per-invitation validity

CSV format: column 1 = email (required); column 2 = comma-separated group ids (optional — quote the cell). An optional header row is recognized. Rows are skipped individually for existing users, duplicate invitations or non-whitelisted domains. Response :

```
{
  "success": true,
  "message": "Imported 2 of 3 users",
  "total": 3,
  "imported": 2,
```

```
"failed": 1,
"success_emails": ["a@x.com", "b@x.com"],
"errors": [ { "email": "c@x.com", "message": "Déjà inscrit" } ],
"parse_errors": ["Ligne 4: email invalide 'not-an-email'"]
}
```

`parse_errors` appears only when some lines could not be parsed; per-row skip messages are localized to the server language.

6.4 — Emails and links

Only three emails exist in the whole backend: address verification (`/#/verify-email?token=...`), password reset (`/#/reset-password?token=...`) and invitation (`/#/accept-invitation?token=...` , batch import only). All links are built from `X-Forwarded-Proto` / `X-Forwarded-Host` (falling back to `Host`), so the fragments land on *your* frontend — implement these three fragment routes if you replace the bundled UI. When SMTP is disabled (the default), links are printed to the server console instead of being sent.

Chapter VII — Administration

The `/api/admin` router covers server-level administration: restarting the backend, inspecting and switching the template-storage backend, and selecting the attachment store. All mutating routes require a JWT with `is_admin` (chapter 2); the errors are the standard `401 {"detail": "Not authenticated"} / 403 {"detail": "Admin access required"}`.

Related administrative surfaces live elsewhere: user account administration under `/api/auth/users` (chapter 2), admin promotion under `/api/users/admins` (chapter 6), store migration execution under `/api/admin/migration` (chapter 9), alerts configuration under `/api/admin/alerts` and diagnostics under `/api/dev` (chapter 11).

7.1 — Restarting the server

A frontend exposes restart for one reason: configuration changes (storage backend, `izibuddy.ini` edits) only take effect on process start.

GET /api/admin/restart/status

Anonymous. Returns the in-memory restart state: `{"pending": false, "countdown": 0}`. Poll this to render a countdown banner.

POST /api/admin/restart

Admin. Body (optional, may be omitted entirely):

Field	Type	Default	Meaning
<code>countdown</code>	int	<code>0</code>	seconds before restart; <code>0</code> = immediate
<code>clear_console</code>	bool	<code>true</code>	clear the server console before re-exec

With a positive countdown the response is `{"success": true, "message": "Server will restart in 30 seconds", "countdown": 30}` and the restart is scheduled (and cancellable). With `countdown: 0` the process re-executes itself immediately — the documented response usually never reaches the client because the connection drops.

If a restart is already scheduled, the endpoint answers HTTP `200` with `{"success": false, "error": "Restart already pending"}`.

The restart is a full process replacement: all in-memory state and open connections (including SSE streams, chapter 10) are lost. A frontend should detect the server coming back by polling a cheap anonymous endpoint (`GET /api/auth/status` or `GET /health`) and then reconnect its event stream.

POST /api/admin/restart/cancel

Admin, no body. {"success": true, "message": "Restart cancelled"} or — again HTTP 200 — {"success": false, "error": "No restart pending"}.

7.2 — Storage backend configuration

IziBuddy's prompt store sits on a pluggable storage abstraction — from per-user JSON files with git history to SQL, NoSQL and object stores. Two endpoints expose the catalogue and the active configuration; a third rewrites it. Switching backends is a two-phase operation: **write the configuration here, then run the data migration** (`/api/admin/migration`, chapter 9) **and restart**.

GET /api/admin/migration/storages

Admin. The full catalogue of supported backends, flagged against the server's current configuration :

```
{
  "storages": {
    "multi-user-with-git": {
      "label": "Multi-User avec Git",
      "fields": ["path", "git_remote", "git_branch"],
      "default_path": "~/.izibuddy/templates.repo",
      "packages": ["gitpython"],
      "category": "filesystem",
      "available": true,
      "enabled": true,
      "current": true,
      "stored_config": { "path": "~/.izibuddy/templates.repo", "git_branch": "main" }
    },
    "mysql": {
      "label": "MySQL/MariaDB",
      "fields": ["host", "port", "database", "user", "password"],
      "default_port": 3306,
      "packages": ["pymysql"],
      "category": "sql",
      "available": true,
      "enabled": false,
      "current": false,
      "stored_config": {}
    }
  },
  "current_storage": "multi-user-with-git"
}
```

The sixteen storage ids: `multi-user-with-git`, `fs-multi-user`, `fs-single-user`, `sqlite`, `mysql`, `postgresql`, `mongodb`, `oracle`, `db2`, `access`, `s3`, `gcs`, `azure`, `minio`, `cockroachdb`, `elasticsearch`. Per entry: `fields` lists which connection fields

the configuration form should show, `enabled` means a configuration section exists in `izibuddy.ini`, `current` marks the active one. Beware: `stored_config` echoes the raw configuration **including stored secrets**.

POST /api/admin/migration/config

Admin. Rewrites the `[database]` section of `izibuddy.ini` — it does **not** restart the server or migrate data. Body :

Field	Type	Re- qui- red	Applies to
<code>storage</code>	string	yes	one of the 16 ids (400 "Unknown storage type: ..." otherwise)
<code>connection</code>	string	no	full connection string
<code>host</code> , <code>port</code> , <code>database</code> , <code>user</code> , <code>password</code>	mixed	no	SQL / NoSQL backends (sensible defaults: <code>localhost</code> , per-storage default port, <code>izibuddy</code>)
<code>path</code>	string	no	filesystem / sqlite / access (falls back to the storage's default path)
<code>git_remote</code> , <code>git_branch</code>	string	no	git storage (branch defaults to <code>main</code>)
<code>bucket</code> , <code>region</code> , <code>access_key</code> , <code>secret_key</code>	string	no	S3 / MinIO
<code>container</code> , <code>connection_string</code>	string	no	Azure
<code>project</code>	string	no	GCS
<code>endpoint</code>	string	no	MinIO

Only fields relevant to the chosen storage are written ; previous `[database]` keys are cleared first. Response :

```
{ "success": true, "message": "Configuration updated to mysql", "storage": "mysql", "co
```

Side effects: a timestamped backup of the ini is written next to it (`izibuddy.ini.YYYYMMDD_HHMMSS.bak`) ; credentials are stored in clear text in the file.

GET /api/admin/migration/config

Admin. The current `[database]` configuration with secrets masked :

```
{ "configured": true, "active": "multi-user-with-git", "path": "~/.izibuddy/templates
```

`{"active": "multi-user-with-git", "configured": false}` when no explicit configuration exists (the default filesystem+git store applies).

7.3 — Attachment storage

Template attachments (chapter 3) have their own pluggable store, independent of the prompt storage backend.

GET /api/admin/attachments/backend

Admin. `{"active": "base64", "supported": ["base64", "sqlite", "filesystem"]}`.

PUT /api/admin/attachments/backend

Admin. Body `{"backend": "sqlite"}` (must be in the supported list, 400 otherwise).
Response `{"success": true, "active": "sqlite"}`.

No automatic migration. Switching the attachment backend affects new saves only; existing attachments stay where they were written (a separate server-side CLI migrates them). Frontends should surface this to the administrator.

7.4 — Administrative flows at a glance

Change of storage backend — the safe sequence a frontend should guide the administrator through :

1. GET /api/admin/migration/storages — render the catalogue and the form fields for the chosen backend.
2. POST /api/admin/migration/config — persist the new configuration.
3. Run the data migration itself (chapter 9): POST /api/admin/migration/export dumps the current store to an archive, then POST /api/admin/migration/import loads it into the newly-configured backend. There is no server-side «test connection» endpoint — the extension's Test button simply polls GET /api/status for reachability.
4. POST /api/admin/restart — bring the server up on the new backend.
5. Poll GET /health until it answers; reconnect SSE.

Emergency restart — POST /api/admin/restart with a countdown, a banner driven by GET /api/admin/restart/status, and POST /api/admin/restart/cancel behind an « abort » button.

Partie 4

Lifecycle and Quality

Chapter VIII — Versioning, Git and Diff

Three services track how templates change over time: **semantic versions** (`/api/versions` — the per-template `0.1`, `0.2`, ... history a user sees in the UI), **git** (`/api/git` — the repository plumbing underneath the default store), and **diff** (`/api/diff` — a stateless text-comparison utility for rendering side-by-side changes).

Authentication is heterogeneous here: version routes use the `X-IziBuddy-User` header (401 without it); the git configuration update requires a JWT admin; everything else in git and diff is anonymous.

8.1 — Semantic versions — `/api/versions`

Every prompt save (chapter 3) records a version entry in a store-wide index; content retrieval reaches into the git history. The wire shape of a version:

```
{
  "format": 1,
  "version": "1.3",
  "sha": "a1b2c3d4",
  "datetime": "2026-07-01 14:32:00",
  "operation": "u",
  "note": "Refonte de la section task",
  "checksum": "e3b0c442"
}
```

- `version` — `"x.y"`; the first version is `0.1`, a minor save bumps `y`, a major save bumps `x` and resets `y` (triggered by `major_version: true` on the prompt update, chapter 3).
- `sha` — the git commit, truncated to 8 characters.
- `operation` — `"c"` create, `"u"` update, `"r"` restore, `"m"` migrated from raw git history.
- `checksum` — 8 hex chars of the content's SHA-256 (line endings normalized). Saving identical content does **not** create a new version.

GET `/api/versions/{template_id}`

All versions of a prompt, oldest first, plus the current number:

```
{
  "success": true,
  "template_id": "code-review",
  "versions": [ { ...version... }, ... ],
  "current_version": "0.2"
}
```

An unknown prompt returns `200` with an empty list and `current_version: null`, not a `404`.

GET /api/versions/{template_id}/{version}

The full prompt JSON as it was at that version, recovered from git (`{"success": true, "version": "0.2", "prompt": { ...stored prompt... }}`). The identity header is used here to locate the user's file. If the recorded SHA is no longer resolvable (history rewritten), the service walks the file's git log and matches versioned or legacy commit messages. Unrecoverable versions return `404` with a localized detail.

Restore is client-side

There is **no restore endpoint**. To restore: fetch the old content with the route above, then save it through the normal prompt update (`PUT /api/templates/update/{id}` with `version_operation: "r"`). This records a new version with operation `"r"` — history is append-only; old versions are never deleted.

Migration from raw git history

- `POST /api/versions/{template_id}/migrate` — converts the file's git history into index entries `0.1`, `0.2`, ... (operation `"m"`). Response `{"success": true, "message": "...", "versions_created": 7}`. A missing file yields `versions_created: 0`, still successful. **Warning: the prompt's existing version list is overwritten** by the migrated list.
- `POST /api/versions/migrate-all` — the same for every prompt of the user; per-template failures are collected in `errors` without aborting: `{"success": true, "migrated": 12, "errors": null, "message": "..."}.`

POST /api/versions/{template_id}/copy

Reserves a copy id for a prompt *at a given version* (body `{"version": "0.9"}`); the new id gets an alphabetical suffix (`code-review_a`, `_b`, ...). Response `{"success": true, "message": "...", "new_template_id": "code-review_a"}`; `404` if the version content cannot be resolved.

***Caveat:** as currently implemented this endpoint only resolves the content and generates the id — it does **not** write the new prompt. The frontend must follow up with `POST /api/templates/create` using the returned id and the fetched historical content.*

8.2 — Git plumbing — /api/git

These routes manage the git repository behind the default file store: initialization, remote, manual commit/push/pull and synchronization settings.

Storage gate. Git only makes sense on the filesystem backend. On DB-backed installs every route except `/status` fails with `409 {"detail": "Git not applicable for this storage backend"}`, and `GET /status` answers `{"success": true, "applicable": false, ...}` — check it first and hide the git UI when `applicable` is false.

Error convention. Operational git failures come back as HTTP 200 with `{"success": false, "error": "..."};` only the gate (409), admin auth (401/403) and validation (400) use HTTP statuses.

GET /api/git/status

```
{
  "success": true,
  "applicable": true,
  "enabled": true,
  "initialized": true,
  "remote": "https://user:token@github.com/acme/templates.git",
  "branch": "main",
  "changes": 3,
  "last_commit": { "hash": "a1b2c3d4", "message": "Update prompt: Code review (2026-0",
  "auth_mode": "token",
  "auto_push": true,
  "pull_strategy": "ff-only"
}
```

`changes` counts dirty files; `auth_mode` is guessed from the remote URL (`token` , `ssh` , `deploy_key` , `none`). Note that `remote` can embed credentials — mask it before display.

Setup and sync

Method & path	Effect
POST /api/git/init	git init + .gitignore + initial commit; idempotent ("Git already initialized")
POST /api/git/remote	add/replace a remote. Body { "url": "...", "name": "origin" } (name optional); enables auto-push
DELETE /api/git/remote/{name}	remove a remote (removing <code>origin</code> disables auto-push); succeeds even if absent
POST /api/git/commit	manual commit. Body { "message": "...", "files": null } (<code>null</code> stages everything); a timestamp is appended to the message. Response merges the commit result: { "success": true, "committed": true, "pushed": true, "push_error": null, "commit_error": null, "sha": "..."} — committed: <code>false</code> with no error means « nothing to commit »

Method & path	Effect
GET /api/git/config	the [git] settings: {"success": true, "auto_push": true, "pull_strategy": "ff-only", "sync_on_startup": true, "sync_on_read": false, "sync_throttle_seconds": 10}
PUT /api/git/config	admin (JWT) . Partial update of the same five keys ; pull_strategy must be ff-only, rebase or merge (400 otherwise). Returns {"success": true, "updated": [... keys...]}

Broken routes to avoid

Five git routes are currently defective against the real git manager and should not be wired into a new frontend :

- POST /api/git/push and POST /api/git/pull — the underlying push/pull **does execute**, but a response-mapping bug makes the reported result meaningless. Both handlers do `success, message = git.push() / git.pull()`, yet the manager returns a **dict**, not a 2-tuple. Unpacking a dict yields its *keys*, so the reported `success/message` are the dict's key names rather than its values, and any result dict that does not have exactly two keys raises `"too many values to unpack (expected 2)"` — returned verbatim as the error. The response therefore reflects the shape of the manager's result dict, never the git outcome. If you need these, treat the response as «operation attempted» and confirm via GET /api/git/status.
- GET /api/git/log, GET /api/git/diff/{commit_hash}, GET /api/git/remotes — they call methods that don't exist (or with the wrong arity) and always return `{"success": false, "error": "..."}.`

In practice a frontend rarely needs them: every prompt mutation already commits and pushes automatically (chapter 3), and per-template history is better served by /api/versions.

8.3 — Text comparison — /api/diff

A single, stateless, anonymous endpoint powering « what changed » views.

POST /api/diff/compare

Body :

Field	Type	Default	Description
text_old	string	required	old text
text_new	string	required	new text
word_mode	bool	false	add word-level spans to modified blocks

The texts are split into blocks (one per line, except fenced code blocks which stay whole), classified (`title`, `code`, `list`, `paragraph`), matched, and **consecutive blocks with the same status are merged**. Response :

```
{
  "blocks": [
    { "id": "block_0", "type": "title", "status": "unchanged",
      "content_old": "# Recette", "content_new": "# Recette",
      "old_spans": null, "new_spans": null },
    { "id": "block_1", "type": "paragraph", "status": "modified",
      "content_old": "Cuire 10 minutes.", "content_new": "Cuire 12 minutes.",
      "old_spans": [
        { "text": "Cuire", "changed": false }, { "text": " ", "changed": false },
        { "text": "10", "changed": true }, { "text": " ", "changed": false },
        { "text": "minutes.", "changed": false }
      ],
      "new_spans": [
        { "text": "Cuire", "changed": false }, { "text": " ", "changed": false },
        { "text": "12", "changed": true }, { "text": " ", "changed": false },
        { "text": "minutes.", "changed": false }
      ] },
    { "id": "block_2", "type": "list", "status": "added",
      "content_old": null, "content_new": "- Servir chaud\n- Saler",
      "old_spans": null, "new_spans": null }
  ]
}
```

`status` is `unchanged`, `modified`, `added` or `removed`; `content_old` is `null` for added blocks and `content_new` `null` for removed ones. The span arrays appear only with `word_mode` on `modified` blocks: alternating word/whitespace tokens flagged `changed`, with newlines as unchanged separators. The only error is FastAPI's 422 for a malformed body.

A typical version-history UI combines the two services: fetch two versions via `/api/versions/{id}/{version}`, then feed both `content` fields to `/api/diff/compare` with `word_mode: true`.

Chapter IX — Scoring and Migration

Two unrelated but equally self-contained services close the lifecycle part: **template-quality scoring** (`/api/scoring` — deterministic structural scoring of a prompt text) and **data migration** (`/api/admin/migration` — exporting and importing the whole store when changing backends).

9.1 — The scoring model

Scoring is pure computation — no auth, no side effects. It analyzes the POO²L structure of a prompt against a **goal** and produces a 0-10 score, a badge and actionable advice. The concepts :

- **Sections** are detected from the POO²L markers: `@@@section name` (required), `@@@section! name` (toggleable, default ON), `@@@section? name` (toggleable, default OFF). Names normalize through French/English aliases to 15 canonical keys: `role`, `context`, `objective`, `task`, `steps`, `constraints`, `exclusions`, `rules`, `examples`, `output_format`, `tone`, `audience`, `system`, `reasoning`, `variables`.
- **Presence is proportional to length**: each section receives a factor 0.0-1.0 based on word or item counts against configured minimums, and counts as *present* when the factor clears its threshold (default 0.5).
- **Goals** describe what the prompt is for; each defines base sections, critical sections and weighted coherence checks between sections. The nine goals: `PRODUIRE`, `ANALYSER`, `EXPLORER`, `CONTRAINDRE`, `AUTOMATISER`, `INCARNER`, `TRANSFORMER`, `EVALUER`, `ORCHESTRER`.
- **Score** = base coverage (weight 6) + coherence (weight 4), clamped to 0-10; `score_min / score_max` bracket the range with all toggleable sections off/on. Badges run from / (very low, or a *forced error* when a critical section is missing or too thin) through    to  and .

9.2 — Scoring routes — `/api/scoring`

GET `/api/scoring/goals`

```
{"success": true, "goals": [{"key": "PRODUIRE", "name": "PRODUIRE"}, ..., {"key": "EVALUER", "name": "ÉVALUER"}]}
```

 — keys are stable identifiers, names are display strings.

POST `/api/scoring/score`

Body :

Field	Type	Default	Notes
<code>text</code>	string	required	prompt text with POO ² L markers

Field	Type	Default	Notes
goal	string	"PRO-DUIRE"	case-insensitive ; unknown values silently fall back to PRODUIRE
standard	string[] null	null	allowed section keys ; null = all 15
optional_overrides	object of bool null	null	force toggleable sections on/off by key
weight_overrides	object null	null	accepted but ignored (legacy)

Response (abridged — every field shown is always present) :

```
{
  "success": true,
  "score": 7.4,
  "score_min": 6.1,
  "score_max": 8.2,
  "score_range": "6.1 → 8.2",
  "base_fraction": 0.81,
  "coherence_fraction": 0.63,
  "badge": "●",
  "badge_tooltip": "Bonne structure. Fiable dans la plupart des cas, encore perfectib",
  "goal": "PRODUIRE",
  "has_forced_error": false,
  "missing_required": [],
  "active_sections": ["task", "context", "objective", "output_format", "constraints",
  "present_sections": ["task", "context", "output_format"],
  "missing_sections": ["objective", "constraints", "rules", "examples"],
  "optional_sections": { "examples": true },
  "coherence_checks": [
    {
      "check_id": "P1",
      "label": "Tâche ↔ Format",
      "mode": "strict_dep",
      "sections": ["task", "output_format"],
      "weight": 2.2,
      "raw_score": 0.95,
      "effective_weight": 5.5,
      "contribution": 5.22,
      "missing_sections": []
    }
  ],
  "failed_checks": ["Tâche ↔ Contraintes", "Contrat (Tâche+Format+Règles)"],
  "dynamic_tooltip": "Bonne structure...\n📊 Couverture : 100%\n➕ À ajouter : constrai",
  "coverage": 1.0
}
```

Reading guide: `goal` echoes the effective goal after fallback; `missing_required` lists critical sections that are absent *or present but too thin* (either forces `has_forced_error: true` and a  badge); `optional_sections` maps each togglable section to its state after overrides; `failed_checks` holds at most two coherence labels; `dynamic_tooltip` is a pre-built multi-line advice string. **Badge tooltips and advice strings are hardcoded French** — localize client-side if needed.

POST /api/scoring/sections

Section detection without scoring. Body `{"text": "..."} . Response:`

```
{
  "success": true,
  "sections": [
    { "key": "task", "raw_name": "TASK", "mode": "required", "is_optional": false,
      "default_included": true, "has_content": true,
      "content_preview": "Analyser le code fourni et produire un rapport..." },
    { "key": "examples", "raw_name": "EXAMPLES", "mode": "optional_off", "is_optional": true,
      "default_included": false, "has_content": true, "content_preview": "Exemple : ..." }
  ],
  "optional_sections": [ ...the is_optional subset... ]
}
```

mode is required, optional_on or optional_off ; content_preview is the first 100 characters. Sections appear in document order and duplicate keys can occur.

POST /api/scoring/regenerate

Rebuilds the prompt text with togglable sections included or excluded — kept sections are copied verbatim, byte-identical.

Parameter binding quirk: `text` is a **query parameter**, while the body is the bare states object:

```
````
POST /api/scoring/regenerate?text=%40%40%40section%3F%20examples%0A... Content-Type: application/json
{ « examples »: false, « steps »: true } ````
```

Sections not listed keep their default; required sections are always kept. Response `{"success": true, "text": "<regenerated prompt>"}`.

```
GET /api/scoring/section-lengths
```

The minimum-length configuration behind the presence factors, for word counters in structuring dialogs :

```
{
 "success": true,
```

```

 "default_threshold": 0.5,
 "sections": {
 "task": { "type": "text", "min_words": 15, "min_items": 2, "min_item_length": 5,
 "steps": { "type": "list", "min_words": 10, "min_items": 3, "min_item_length": 4,
 }
 }
}

```

`type` is `text` (factor = words/min\_words) or `list` (geometric mean of item count and item length factors), capped at 1.0.

### 9.3 — Data migration — `/api/admin/migration`

Where chapter 7 configured *which* backend the server should use, this router moves the *data*: full-store export, validation and import. All seven routes require **system admin**, authenticated by either the `X-IziBuddy-User` header or a Bearer JWT ( 401 "Authentication required" / 403 "System admin access required" ).

Operations return a common result: counts ( `users`, `groups`, `templates`, `history_entries` ), errors, warnings and `duration_seconds`. Export files live in a **temporary directory** on the server — they are transfer artifacts, not backups.

#### **GET** `/api/admin/migration/status`

Current backend, store statistics and the five most recent exports :

```

{
 "success": true,
 "current_backend": { "type": "File system/Git", "location": "/home/user/.izibuddy/t",
 "stats": { "users": 14, "groups": 3, "templates": 220, "system_admins": 2 },
 "available_exports": [
 { "filename": "izibuddy_export_20260712_101500.json.gz", "size_bytes": 34567, "cr
]
}

```

The `location` field is the storage's on-disk path ( `StorageInfo.path` ); it is `null` on DB-backed backends that have no filesystem location.

#### **POST** `/api/admin/migration/export`

Body (all optional): `format` ( "json" default, or "sqlite" ), `include_history` (default true), `include_templates` (default true), `compress` (default true — gzip for JSON).  
Response :

```

{
 "success": true,
 "filename": "izibuddy_export_20260712_103000.json.gz",
 "stats": { "users": 14, "groups": 3, "templates": 220, "history_entries": 1830 },
 "duration_seconds": 2.41,

```

```
"download_url": "/api/admin/migration/download/izibuddy_export_20260712_103000.json.g
}
```

Failures are 500 with the joined error list as detail.

#### GET /api/admin/migration/download/{filename}

Streams the export file as an attachment ( application/gzip , application/x-sqlite3 or application/json ). Only files inside the export directory are served; 404 otherwise.

#### POST /api/admin/migration/validate

Dry run. multipart/form-data with one file field. Response {"success": true, "valid": true, "stats": { ... }, "errors": []} — an invalid file is still HTTP 200 with valid: false and populated errors.

#### POST /api/admin/migration/import

The real import. multipart/form-data :

| Field          | Type            | Default  | Notes                                          |
|----------------|-----------------|----------|------------------------------------------------|
| file           | upload          | required | .db → SQLite import; anything else → JSON(.gz) |
| clear_existing | bool form field | false    | <b>wipes all current data first</b> when true  |

Response {"success": true, "stats": { ... }, "warnings": ["User 'bob' already exists, skipped"], "duration\_seconds": 4.02} (warnings capped at ten). Treat clear\_existing as a destructive action deserving explicit confirmation in your UI.

### Supporting routes

- DELETE /api/admin/migration/exports/{filename} — remove an export file ( 404 if absent).
- GET /api/admin/migration/backends — a static catalogue of the twelve migratable backends ( {"id": "sqlite", "name": "SQLite", "description": "Single file database, zero config"} , ...) for building the selection UI.

### The full backend-switch flow

1. On the source install: POST /export → GET /download/{filename}.
2. Reconfigure the backend ( /api/admin/migration/config , chapter 7) and restart.
3. On the new backend: POST /validate with the file → review counts and errors → POST /import (with clear\_existing as appropriate).
4. GET /status to confirm the counts; delete the export file.

# Partie 5

## *Platform Services*



## Chapter X — Real-Time Events, Status and Health

The backend pushes real-time notifications to connected frontends over **Server-Sent Events** (SSE), and exposes a handful of status endpoints for liveness and introspection. Together they let a frontend show « server is restarting » overlays, detect missed restarts after a reconnection, and verify what it is connected to.

### 10.1 — The SSE stream — `GET /api/events/stream`

A long-lived `text/event-stream` connection carrying server-to-client notifications: planned shutdowns, and « server is back » signals.

**Identity is client-declared.** The endpoint has no authentication. Because the browser `EventSource` API cannot set headers, identity travels in query parameters (headers work too, but query parameters win) :

| Query param                  | Header alternative                       | Meaning                                     |
|------------------------------|------------------------------------------|---------------------------------------------|
| <code>user</code>            | <code>X-IziBuddy-User</code>             | username (defaults to <code>anonym</code> ) |
| <code>sysadmin=true</code>   | <code>X-IziBuddy-Sysadmin: true</code>   | treat the client as a system admin          |
| <code>groupadmin=true</code> | <code>X-IziBuddy-GroupAdmin: true</code> | treat the client as a group admin           |

The role flags affect **message wording only** (admins see who initiated a shutdown; ordinary users see a masked name) — they gate no data beyond that attribution.

**Wire format.** Every message is an unnamed SSE event: `data: <JSON>\n\n`. The server never sets `event:` or `id:` fields, so listen with `onmessage` (not named-event listeners), and there is **no Last-Event-ID replay**. Every 30 seconds of silence the server emits a comment heartbeat (`: heartbeat`) — invisible to `onmessage`, but it keeps proxies from killing the connection. Response headers include `X-Accel-Buffering: no` to defeat nginx buffering.

**Event catalogue** (the `type` field of the JSON payload) :

- `connected` — handshake, sent once per (re)connection: `{"type": "connected", "message": "Connecté au serveur"}` (the only event without a `timestamp`).
- `shutdown` — broadcast before a planned stop or restart: `{"type": "shutdown", "timestamp": "2026-07-12T09:41:00.123456", "initiated_by": "admin", "message": "..."}.` Admin clients receive the real initiator; ordinary clients see `initiated_by: "sysadmin"`.
- `ready` — broadcast when the server is back: `{"type": "ready", "timestamp": "...", "message": "..."}.` (no `initiated_by` field; only the message differs by role).

- `restart` — defined in the protocol and worth handling defensively, but no current code path emits it.

Timestamps are server-local ISO strings (no `Z` suffix).

**Reconnection.** `EventSource` auto-reconnects (~3 s default; no `retry:` directive is sent). Events broadcast while disconnected are **lost** — after every reconnect, poll `GET /api/events/status` and inspect `last_shutdown` to detect a missed shutdown/ready cycle.

A minimal client :

```
const es = new EventSource(`/api/events/stream?user=${encodeURIComponent(username)}&sys
es.onmessage = (e) => {
 const msg = JSON.parse(e.data); // {type, timestamp?, message, initiated_by?}
 switch (msg.type) {
 case 'connected': /* mark online */ break;
 case 'shutdown':
 case 'restart': /* show a blocking "server stopping" overlay */ break;
 case 'ready': /* dismiss the overlay, refresh data */ break;
 }
};
// After a reconnect, poll /api/events/status: a non-null last_shutdown
// means a shutdown happened and no ready has been broadcast since.
```

**Presence.** Each open stream counts as one connected client (one per tab). There are no join/leave broadcasts; presence surfaces only as the count in `/api/events/status` and in the diagnostic report's `frontends_connected`.

## 10.2 — Event support endpoints

### `GET /api/events/status`

Anonymous, poll-able :

```
{
 "connected_clients": 3,
 "last_shutdown": { "initiated_by": "admin", "timestamp": "2026-07-12T09:41:00.123456" }
}
```

`last_shutdown` is `null` in normal operation; it is set by a shutdown notification and cleared back to `null` by the next `ready` — the reconnect-recovery signal described above.

### **POST /api/events/notify/shutdown**

Broadcasts a `shutdown` event to every connected client and records `last_shutdown`. Requires the `X-IziBuddy-User` header — but note it is a required *header parameter*, so a missing header yields `422`, not `401`, and **there is no admin gate**: any caller with the header can broadcast the banner. Response `{"success": true, "notified": 3}`. (The server also calls this internally before Lab auto-reload restarts.)

### **POST /api/events/notify/ready**

Broadcasts `ready` and clears `last_shutdown`. Header optional. Response `{"success": true, "notified": 3}`.

## **10.3 — Status and introspection**

Four anonymous endpoints without a common prefix:

### **GET /api/status**

Liveness probe: `{"status": "running", "version": "1.0", "mode": "async"}`.

### **GET /api/queue/stats**

A hard-coded stub retained for compatibility — always `{"success": true, "stats": {"running": false}, "active_tasks": []}`. Do not build features on it.

### **GET /api/routes**

Live route introspection, useful for tooling and contract tests:

```
{
 "routes": [
 { "path": "/api/analytics/summary", "methods": ["GET"] },
 { "path": "/api/analytics/track", "methods": ["POST"] }
]
}
```

### **GET /api/routefile**

The same introspection rendered as a self-contained **HTML page** — a searchable table of every route with method badges, source files, and inline inputs to try parameterized paths. Developer tooling, not a JSON API.

## **10.4 — Health — GET /health**

The application-level health endpoint (chapter 1) belongs in the same toolbox:

```
{
 "status": "healthy",
 "components": {
```

```

 "config_manager": true,
 "templates_manager": true,
 "analyzer": true,
 "concatenator": true,
 "backup_manager": true
 },
 "engine": { "capable": true, "source": ".../libpool.so", "detail": "..." }
}

```

`status` is always `"healthy"` when the process answers — real readiness lives in `components` (which managers initialized) and `engine.capable`. A `false` engine capability means the native POO<sup>2</sup>L binary is stale: renders would come back empty; surface a prominent warning and point the administrator at rebuilding the engine.

## 10.5 — Putting it together: restart-aware frontends

The full resilience recipe used by the bundled web app:

1. Open the SSE stream at boot with the username and role flags.
2. On `shutdown`, show a blocking overlay (« server is restarting... »).
3. On `ready`, dismiss the overlay and refresh volatile data.
4. On every `EventSource` reconnect, poll `/api/events/status`; if `last_shutdown` is non-null, treat it as a missed `shutdown` (overlay) and wait for `ready`; if the server answers normally and `last_shutdown` is null, resume silently.
5. In development (`--reload`), also poll `GET /api/config` for the `boot_id` (chapter 12): `uvicorn`'s reload hands the socket to the new process, so the SSE connection may survive a code swap that the `boot_id` reveals.

## Chapter XI — Analytics, Alerts and Diagnostics

Three observability services: **usage analytics** (`/api/analytics` — tracking and dashboards), **administrative alerts** (`/api/admin/alerts` — threshold-based operational alerts), and **diagnostics** (`/api/dev` — support reports). Analytics and alerts authenticate with the `X-IziBuddy-User` header (plus a system-admin check where noted); the diagnostic routes are the one observability surface using real JWT Bearer auth.

### 11.1 — Two analytics subsystems

The analytics API sits on **two independent stores** — understanding this prevents a classic confusion :

| Subsystem             | Storage                         | Event names                                                                                                                                                            | Serves                                                                                                                            |
|-----------------------|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Raw event log         | append-only JSONL file          | free-form strings ( <code>instantiate</code> , <code>view</code> , <code>edit</code> , <code>create</code> , <code>delete</code> , <code>restore</code> by convention) | <code>/track</code> , <code>/summary</code> , <code>/templates/{id}</code> , <code>/export/csv</code> , <code>/export/json</code> |
| Aggregated dashboards | daily JSONL + SQLite aggregates | a fixed event-type enum ( <code>template_view</code> , <code>template_create</code> , ..., <code>user_login</code> , <code>error</code> , ...)                         | all <code>/dashboard/*</code> and <code>/export/dashboard/*</code>                                                                |

An event recorded through `POST /track` **never** appears in the dashboards, and dashboard metrics are fed by internal instrumentation, not by `/track`. The two taxonomies also differ (`instantiate` vs `template_instantiate`).

### 11.2 — Tracking and raw queries

#### `POST /api/analytics/track`

Records one usage event (any user). Body :

| Field                    | Type            | Required | Notes                                 |
|--------------------------|-----------------|----------|---------------------------------------|
| <code>event_type</code>  | string          | yes      | free-form ; conventional values above |
| <code>template_id</code> | string          | yes      |                                       |
| <code>version</code>     | string   null   | no       | prompt version used                   |
| <code>preset</code>      | string   null   | no       | preset applied                        |
| <code>sections</code>    | string[]   null | no       | optional sections enabled             |

Response `{"success": true}` — note that a storage I/O failure yields `200 {"success": false}`, not an error status.

#### **GET /api/analytics/summary?period\_days=30**

Global rolling-window summary (any user; `period_days` 1-365):

```
{
 "success": true,
 "period_days": 30,
 "total_events": 412,
 "unique_templates": 37,
 "unique_users": 9,
 "event_types": { "instantiate": 350, "view": 40, "edit": 22 },
 "top_templates": [{ "template_id": "a1b2c3d4", "count": 88 }]
}
```

#### **GET /api/analytics/templates/{template\_id}?period\_days=30**

Per-template statistics, counting **only instantiate events**:

```
{
 "success": true,
 "template_id": "a1b2c3d4",
 "total_instantiations": 88,
 "unique_users": 5,
 "presets_used": { "client-a": 60, "client-b": 28 },
 "versions_used": { "2.3": 70, "2.2": 18 },
 "sections_used": { "RGPD": 44, "CTA": 30 },
 "daily_counts": { "2026-07-10": 12, "2026-07-11": 9 }
}
```

Unknown ids return zero-filled stats, not 404.

## **Raw exports**

`GET /api/analytics/export/csv` and `GET /api/analytics/export/json` — filtered file downloads of the raw event log. Filters (all optional query params): `start_date`, `end_date` (ISO bounds), `template_ids`, `event_types` (comma-separated). The CSV has columns `date,template_id,user,event_type,version,preset` with **no quoting or escaping**; the JSON is the compact stored form ( `ts`, `type`, `prompt`, `user`, optional `v` / `preset` / `sections` ).

**Privacy note**: these raw exports require only the identity header — they are not admin-gated, and contain the full cross-user activity log. Deployments that care should front them with a proxy rule.

## 11.3 — Dashboards

All dashboards take `days` (default 30, max 365) and read the aggregated store. `503 {"detail": "Analytics not initialized"}` means the tracker was not initialized at startup.

**GET /api/analytics/dashboard/system — system admin**

```
{
 "success": true,
 "period_days": 30,
 "total_events": 5120,
 "active_users": 14,
 "templates_created": 63,
 "templates_edited": 240,
 "templates_instantiated": 1830,
 "logins": 402,
 "failed_logins": 12
}
```

**GET /api/analytics/dashboard/groups — system admin**

Per-group summaries sorted by activity: `{"success": true, "period_days": 30, "groups": [{"group_id": "dev", "group_name": "Development", "total_members": 8, "active_members": 6, "total_events": 1210, "templates_shared": 14}], "total_groups": 1}`.

**GET /api/analytics/dashboard/group/{group\_id}?days=&include\_members=true**

System admin **or admin of that group**. Full group dashboard: totals, `top_templates` (`{id, name, count, unique_users}`), `top_members` (`{username, events}`), ascending `daily_trend` (`{date, events, active_users}`), and — with `include_members=true` — a `member_stats` array (`{username, total_events, templates_created, templates_edited, templates_instantiated, logins}`) sorted by activity.

*`top_templates` and `top_members` always cover the **current calendar month**, whatever days says — label them accordingly in your UI.*

### Per-user dashboards

- **GET /api/analytics/dashboard/user/{username}** — that user or a system admin (403 "Access denied" otherwise): `{"success": true, "username": "alice", "period_days": 30, "total_events": 300, "templates_created": 4, "templates_edited": 20, "templates_instantiated": 150, "logins": 25}`.
- **GET /api/analytics/dashboard/me** — the same for the caller, no admin logic.

## Dashboard report exports

`GET /api/analytics/export/dashboard/system?days=&format=csv|xlsx|pdf` (system admin) and `GET /api/analytics/export/dashboard/group/{group_id}?...` (system or group admin) download formatted reports. CSV embeds `## Summary / ## Details / ## Daily Trend` sections; `xlsx` and `pdf` require optional server dependencies — expect `501 {"detail": "Export format not available: ..."}` when they are missing, and `422` for other format values.

### 11.4 — Administrative alerts — `/api/admin/alerts`

Threshold-based operational alerts, persisted server-side and dispatched to configured channels (log always; email and webhook when configured). All routes require **system admin**; `503 {"detail": "Alert manager not initialized"}` when the subsystem is off.

Alert types: `error_rate`, `failed_login`, `brute_force`, `slow_requests`, `disk_space`, `user_locked`, `security_event`, `system_error`, `custom`. Severities: `info`, `warning`, `critical`. Each type has a threshold within a sliding window and a cooldown (e.g. `failed_login` : 5 events in 15 min, warning, 30 min cooldown; `brute_force` : 20 in 5 min, critical), all overridable in the server's `[alerts]` configuration.

#### `GET /api/admin/alerts`

Paginated list with filters — `alert_type`, `acknowledged` (`true` / `false` / `absent` = all), `page` ( $\geq 1$ ), `page_size` (1-200):

```
{
 "success": true,
 "alerts": [
 {
 "id": "3f8c1e2a-...",
 "alert_type": "failed_login",
 "severity": "warning",
 "message": "Multiple failed login attempts for user bob: 5 attempts",
 "data": { "user": "bob" },
 "created_at": "2026-07-12T07:55:02.412345Z",
 "acknowledged": false,
 "acknowledged_by": null,
 "acknowledged_at": null
 }
],
 "total": 23,
 "page": 1,
 "page_size": 50
}
```

`total` counts all matches, not just the page ; ordering is newest first.

## The rest of the surface

| Method & path                                      | Effect                                                                                                                              |
|----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| GET /api/admin/alerts/recent?limit=10              | most recent <b>unacknowledged</b> alerts, trimmed shape ( id , alert_type , severity , message , created_at ) — the badge/bell feed |
| POST /api/admin/alerts/{alert_id}/acknowledge      | acknowledge one (optional body { "note": "..." }, currently unused). 404 when unknown <i>or already acknowledged</i>                |
| POST /api/admin/alerts/acknowledge-all?alert_type= | acknowledge everything (optionally one type): { "success": true, "message": "Acknowledged 12 alerts", "count": 12 }                 |
| GET /api/admin/alerts/config                       | { "success": true, "enabled": true, "alert_types": [...], "channels": ["LogChannel", "EmailChannel"] }                              |
| GET /api/admin/alerts/stats?days=30                | counters: { "total", "acknowledged", "unacknowledged", "by_type": {...}, "by_severity": {...} }                                     |
| POST /api/admin/alerts/test                        | fires a real security_event (severity info) through every channel — the pipeline check                                              |

## 11.5 — Diagnostics — /api/dev

JWT Bearer authenticated. Two admin-only routes and one for any user.

### GET /api/dev/diagnostic — admin

The full support report as JSON: server block (version, uptime, Python version, `frontends_connected` — the live SSE count, package versions), system block (OS, architecture, IPs, `environment`: `native/docker/wsl2`, CPU and memory with `ok/warning/critical` statuses), storage block (type, path, size, git state), user/group/prompt counts, content counts (collections, libraries, projects, folders), version-index statistics, per-component health, disk usage, the last ten error-log lines, configuration status — **and the raw configuration file including secrets**, which is why this is strictly admin-only.

### GET /api/dev/diagnostic/user — any authenticated user

The sanitized variant (the « 10 clicks on the version number » easter egg): server name/version/uptime plus the caller's own public profile. No paths, no secrets, no other users.

### GET /api/dev/diagnostic/report — admin

The same data as the full diagnostic, rendered as one copy-pasteable ASCII text report: { "report": "====...\n IziBuddy System Diagnostic Report\n..." } — ideal for a « copy support report » button.

## 11.6 — Security posture of this surface

Worth restating for anyone deploying beyond a trusted network: the identity header is an assertion, not a credential (chapter 2). Concretely, on this surface: the raw analytics exports are reachable by any user; the SSE role flags are self-declared; and the shut-down-notification endpoint has no admin gate. The genuinely sensitive routes (full diagnostics with secrets, alert management, dashboards) do check admin status — but for the header-based ones the asserted username is what is being checked. Front the API with real authentication at the proxy level if any of this matters in your deployment.

## Chapter XII — Configuration, Tasks and Licensing

The last services a frontend touches: the **runtime configuration** endpoints (`/api/config` — including the live-reload contract every development frontend should implement) and **licensing** (`/api/license` and the payment-provider webhooks under `/api/webhooks`). A fourth, the **background task queue** (`/api/tasks`), is a desktop-era leftover no longer mounted on the web backend — covered at the end for the record.

### 12.1 — Configuration — `/api/config`

No per-user auth; every route depends on the config manager singleton and answers `503 {"detail": "Config manager not initialized"}` while the server is still starting — a frontend booting in parallel with the backend should treat 503 here as «retry shortly».

#### **GET** `/api/config`

The effective configuration snapshot plus the two live-reload keys:

```
{
 "success": true,
 "config": {
 "server": { "port": 8765, "workers": 4, "host": "127.0.0.1", "alternative_port": 8766 },
 "backups": { "backup_count": 10, "backup_compression": true, "archive_after_data": 1000000, "archive_after_versions": 50, "auto_archive": true, "purge_archive": true },
 "processing": { "max_tokens_summary": 2000, "max_tokens_concat": 8000, "token_estimate": 1000000 },
 "logging": { "log_level": "INFO", "log_max_size": 10485760, "log_backup_count": 10 },
 "ui": { "check_status_interval": 5000, "queue_stats_interval": 2000 },
 "features": { "allow_no_scoring": true }
 },
 "live_reload": false,
 "boot_id": "9f2c1e6a8b3d4e5f9a7b6c5d4e3f2a1b"
}
```

(The `config.*` values mirror the server's settings at runtime; the two `ui.*` intervals are currently hard-coded to 5000/2000 ms.)

**The live-reload contract.** `live_reload` is `true` only when the server runs in a development mode (`--reload` or as a Lab worker). `boot_id` is a UUID fixed per server process — any restart yields a new one. A development frontend should poll `GET /api/config` every couple of seconds while `live_reload` is true and **reload itself when `boot_id` changes**. Polling, not SSE, is the mechanism here because `uvicorn's --reload` hands the listening socket to the new worker: existing SSE connections never drop, so the client would otherwise never notice the restart. In production `live_reload` is `false` and no polling should happen.

## POST /api/config

Partial update — only fields present in the body are applied :

| Field                                                                                                                                           | Type                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| <code>workers</code> , <code>port</code>                                                                                                        | int                                                                                        |
| <code>backup_count</code> , <code>archive_after_days</code> , <code>archive_after_versions</code> ,<br><code>purge_archives_after_months</code> | int                                                                                        |
| <code>auto_archive</code>                                                                                                                       | bool                                                                                       |
| <code>max_tokens_summary</code> , <code>max_tokens_concat</code>                                                                                | int                                                                                        |
| <code>log_level</code>                                                                                                                          | string                                                                                     |
| <code>check_status_interval</code> , <code>queue_stats_interval</code>                                                                          | int — <b>accepted but never applied</b> (the GET returns the hard-coded values regardless) |

Changes mutate the in-process settings and are persisted through the config manager where supported. Response `{"success": true, "message": "..."} (message localized to the server language).`

## Exclusion patterns and reset

- GET `/api/config/exclude-patterns` — file/directory exclusion patterns used by the concatenation/analysis features: `{"success": true, "patterns": {"dirs": ["node_modules", ".git", ...], "files": ["*.pyc", ...], "patterns": [...]}}`.
- POST `/api/config/exclude-patterns` — replaces them; the body is a raw object of the same shape (no Pydantic model). On a config manager that lacks the capability the call is a silent no-op that still reports success.
- POST `/api/config/reset` — restores the in-process defaults (and the manager's, when supported). No body.

## 12.2 — Background tasks — `/api/tasks`

A small worker-pool queue for long-running jobs, ported from the desktop application (`src/workers/task_queue.py` — « Port depuis `TaskQueue.pm` ») and still carrying its GUI-message plumbing. It was never adopted by the web backend : the queue is instantiated nowhere, no frontend or extension calls it, and the analyze / concatenate operations it was meant to run happen synchronously through the managers instead. The router is therefore **no longer mounted** — `create_app()` does not `include_router(tasks.router)`, so **every `/api/tasks/*` path returns 404 (unknown route)**, not a 503. The queue module stays in the tree for a future need ; nothing on the web API exposes it today.

The table below documents that legacy design for reference only; none of it is reachable on the running server. The essential lifecycle fact of the design: **the queue only tracks running tasks** — a finished task disappears, and there is no history endpoint.

| Method & path            | Effect (legacy design, not exposed)                                                                                                                                                                                                                                           |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| POST /api/tasks          | enqueue. Body <code>{"task_type": "...", "params": {...}}</code> . Response <code>{"success": true, "task_id": "task_a1b2c3d4_1720795200123", "data": {"message": "Task ... created"}, "error": null}</code>                                                                  |
| GET /api/tasks           | currently executing tasks: <code>{"success": true, "tasks": [{ "id", "type", "project", "duration"}]}</code> (duration in seconds since start)                                                                                                                                |
| GET /api/tasks/stats     | queue statistics: <code>{"success": true, "stats": {"pending", "active", "workers", "processed", "errors", "running", "uptime"}}</code>                                                                                                                                       |
| GET /api/tasks/{task_id} | poll one task. Found → <code>{"success": true, "task_id", "data": {... same as list entry...}, "error": null}</code> . <b>404 {"detail": "Task not found"}</b> means « <b>unknown or already finished</b> » — a poller that created the task must interpret 404 as completion |

## 12.3 — Licensing — /api/license

IziBuddy uses a freemium model backed by the Lemon Squeezy platform. Plans: `free-mium` (template-count limit) and `pro` (unlimited). License types: `none`, `one_time`, `subscription`; statuses: `active`, `cancelled`, `expired`, `inactive` — the effective plan is `pro` only while the status is `active`.

### GET /api/license/status

Any user (identity header). The payload drives both the plan badge and the create-template gating:

```
{
 "plan": "pro",
 "plan_name": "Pro",
 "type": "subscription",
 "template_limit": null,
 "template_count": 210,
 "can_create": true,
 "status": "active",
 "expires_at": "2026-12-01T00:00:00.000000Z",
 "customer_email": "owner@company.com",
 "product_name": "IziBuddy Pro"
}
```

On freemium, `template_limit` is a number (e.g. 10) and `can_create = template_count < template_limit`; enforce it optimistically in the UI, the backend enforces it on save regardless.

## Admin routes

| Method & path                          | Effect                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| POST /api/license/activate             | body <code>{"license_key": "..."} (≥ 10 chars, else 400 "Invalid license key format")</code> . The backend calls the Lemon Squeezy activation API and, on success, stores the license: <code>{"success": true, "plan": "pro", "type": "one_time", "expires_at": null}</code> . Activation failures return 400 with the provider's message |
| POST /api/license/deactivate           | resets the local license to freemium. <b>Local-only</b> — the provider-side instance seat is not released                                                                                                                                                                                                                                 |
| GET /api/license/portal                | <code>{"url": "https://app.lemonsqueezy.com/my-orders"}</code> when a customer id is on file; 404 <code>{"detail": "No active subscription"}</code> otherwise                                                                                                                                                                             |
| GET /api/license/checkout/{variant_id} | <code>{"url": "https://izibuddy.lemonsqueezy.com/checkout/buy/123456"}</code> — the purchase link to open                                                                                                                                                                                                                                 |

The license also revalidates periodically against the provider; a *failed* validation marks it expired, while network errors leave it untouched.

## Webhooks — /api/webhooks

POST /api/webhooks/lemonsqueezy — the payment-provider callback (no user auth; HMAC-SHA256 signature via the X-Signature header, verified against the configured secret). Handled events update the local license: orders and subscription lifecycle events flip the plan between `pro` and `freemium` and maintain `expires_at`; unknown events are logged and acknowledged. Response `{"received": true, "event": "subscription_created"}`; errors are 400 (bad JSON / missing event name) and 401 (missing or invalid signature).

**Fail-closed signature check:** a missing X-Signature header is rejected with 401 exactly like an invalid one — an unsigned request is never processed. When no webhook secret is configured (Lemon Squeezy disabled for the initial release) the check fails closed and every webhook is refused.

GET /api/webhooks/lemonsqueezy/test — anonymous reachability probe: `{"status": "ok", "message": "Webhook endpoint is reachable"}` — useful when configuring the webhook URL at the provider.

## 12.4 — What a frontend should wire up

- **Boot**: GET `/api/config` once for feature flags ( `allow_no_scoring` ) and the live-reload decision ; retry on 503.
- **Development**: the `boot_id` poll while `live_reload` is true.
- **Long jobs**: there is no server-side job queue to build on — the `/api/tasks` router is not mounted (any `/api/tasks/*` path is a plain 404 ), and the analyze / concatenate operations run synchronously. Call the relevant endpoint and await its response rather than polling a queue.
- **Plan awareness**: GET `/api/license/status` at boot and after any activation flow ; disable creation UI when `can_create` is false, with an upgrade path through `checkout` → payment → webhook → refreshed status.



# Appendices



## Appendix A — Complete Endpoint Index

Every live route of the backend, grouped by router, with its authentication requirement and the chapter that documents it. Legend for the *Auth* column :

| Code        | Meaning                                             |
|-------------|-----------------------------------------------------|
| —           | anonymous                                           |
| JWT         | valid Bearer token ( Authorization: Bearer )        |
| JWT-adm     | Bearer token + system admin                         |
| hdr         | X-IziBuddy-User header (unverified identity)        |
| hdr-adm     | header + system-admin check                         |
| hdr/JWT-adm | header <b>or</b> Bearer token, + system-admin check |
| loop        | loopback clients only (Lab)                         |

Routes marked  $\triangle$  are documented as defective or dead in their chapter — check before wiring them in.

### A.1 — Application level (chapter 1)

| Me-<br>thod | Path            | Auth | Purpose                                             |
|-------------|-----------------|------|-----------------------------------------------------|
| GET         | /               | —    | serve the bundled web app                           |
| GET         | /help           | —    | serve the help page                                 |
| GET         | /dev/diagnostic | —    | serve the diagnostic page (admin check client-side) |
| GET         | /health         | —    | component + engine capability probe                 |
| GET         | /api/info       | —    | API identity                                        |
| GET         | /docs , /redoc  | —    | generated OpenAPI documentation                     |

## A.2 — Authentication — `/api/auth` (chapter 2)

| Method | Path                                                   | Auth      | Purpose                                                           |
|--------|--------------------------------------------------------|-----------|-------------------------------------------------------------------|
| POST   | <code>/api/auth/register</code>                        | —         | open registration (first user becomes admin)                      |
| POST   | <code>/api/auth/login</code>                           | —         | login (username or email), returns JWT                            |
| GET    | <code>/api/auth/me</code>                              | JWT       | current profile                                                   |
| POST   | <code>/api/auth/logout</code>                          | —         | no-op (logout is client-side)                                     |
| POST   | <code>/api/auth/change-password</code>                 | JWT       | change own password                                               |
| GET    | <code>/api/auth/check</code>                           | —         | non-throwing session probe                                        |
| GET    | <code>/api/auth/status</code>                          | —         | bootstrap ( <code>has_users</code> , <code>show_register</code> ) |
| GET    | <code>/api/auth/config/registration</code>             | —         | registration policy for the UI                                    |
| GET    | <code>/api/auth/users</code>                           | JWT-admin | list accounts                                                     |
| PUT    | <code>/api/auth/users/{username}</code>                | JWT-admin | update email / <code>is_admin</code> / <code>is_active</code>     |
| DELETE | <code>/api/auth/users/{username}</code>                | JWT-admin | delete an account                                                 |
| POST   | <code>/api/auth/register/initiate</code>               | —         | self-registration : send verification email                       |
| GET    | <code>/api/auth/register/validate/{token}</code>       | —         | validate verification token                                       |
| POST   | <code>/api/auth/register/complete</code>               | —         | create account from token                                         |
| POST   | <code>/api/auth/password-reset/initiate</code>         | —         | send reset email (always 200)                                     |
| GET    | <code>/api/auth/password-reset/validate/{token}</code> | —         | validate reset token                                              |
| POST   | <code>/api/auth/password-reset/complete</code>         | —         | set new password                                                  |

### A.3 — User settings — `/api/users` (chapter 6)

| Method | Path                                         | Auth      | Purpose                                                        |
|--------|----------------------------------------------|-----------|----------------------------------------------------------------|
| GET    | <code>/api/users</code>                      | —         | usernames with a settings directory                            |
| GET    | <code>/api/users/me</code>                   | hdr       | both settings bundles                                          |
| PUT    | <code>/api/users/me/web</code>               | hdr       | update web settings (presets, features, themes)                |
| PUT    | <code>/api/users/me/extension</code>         | hdr       | update extension settings                                      |
| GET    | <code>/api/users/me/is-admin</code>          | hdr       | admin check                                                    |
| GET    | <code>/api/users/{username}</code>           | —         | user info + defaults (never 404)                               |
| POST   | <code>/api/users/{username}</code>           | —         | provision settings directories                                 |
| GET    | <code>/api/users/{username}/web</code>       | —         | web settings (bare object)                                     |
| PUT    | <code>/api/users/{username}/web</code>       | —         | update web settings by path                                    |
| GET    | <code>/api/users/{username}/extension</code> | —         | extension settings                                             |
| PUT    | <code>/api/users/{username}/extension</code> | —         | update extension settings                                      |
| GET    | <code>/api/users/settings/backend</code>     | —         | global backend settings                                        |
| PUT    | <code>/api/users/settings/backend</code>     | —         | update global backend settings                                 |
| GET    | <code>/api/users/admins</code>               | —         | ⚠ shadowed by <code>/api/users/{username}</code> — unreachable |
| POST   | <code>/api/users/admins/{username}</code>    | hdr-admin | promote admin (bootstrap : first self-appointment)             |
| DELETE | <code>/api/users/admins/{username}</code>    | hdr-admin | demote admin                                                   |

### A.4 — Templates — `/api/templates` (chapters 3-4)

All JWT. Failures usually HTTP 200 `{"success": false}`.

| Method | Path                                                | Purpose                                |
|--------|-----------------------------------------------------|----------------------------------------|
| GET    | /api/templates/list                                 | full listing (+trash option)           |
| GET    | /api/templates/get/{template_id}                    | one prompt, editing shape              |
| GET    | /api/templates/get/{template_id}/inheritance-chain  | OO chain, root first                   |
| GET    | /api/templates/get/{template_id}/inherited-sections | ancestor sections                      |
| POST   | /api/templates/create                               | create                                 |
| PUT    | /api/templates/update/bulk                          | bulk whitelist patches, one commit     |
| PUT    | /api/templates/update/{template_id}                 | update (content, params, versioning)   |
| DELETE | /api/templates/delete/{template_id}                 | soft delete (locked refused)           |
| POST   | /api/templates/share/{template_id}                  | ⚠ legacy — not_supported               |
| POST   | /api/templates/unshare/{template_id}                | ⚠ legacy — not_supported               |
| POST   | /api/templates/duplicate/{template_id}              | ⚠ legacy — not_supported               |
| GET    | /api/templates/trash/count                          | trash size                             |
| GET    | /api/templates/trash/list                           | trash contents                         |
| POST   | /api/templates/restore/{template_id}                | restore (optional new primary tag)     |
| DELETE | /api/templates/purge/{template_id}                  | hard delete one                        |
| DELETE | /api/templates/purge-selection                      | hard delete many (JSON body on DELETE) |
| DELETE | /api/templates/purge-all                            | empty trash                            |
| GET    | /api/templates/scope/list                           | scope-filtered listing                 |
| GET    | /api/templates/groups                               | caller's groups (context selector)     |
| GET    | /api/templates/hidden                               | hidden ids (user + groups)             |

| Method | Path                                                | Purpose                                                      |
|--------|-----------------------------------------------------|--------------------------------------------------------------|
| POST   | /api/templates/hidden/{template_id}                 | toggle hidden                                                |
| PUT    | /api/templates/hidden/{template_id}                 | set hidden explicitly                                        |
| GET    | /api/templates/search?q=                            | full-text search                                             |
| GET    | /api/templates/render/{template_id}                 | render, system variables only                                |
| POST   | /api/templates/render/{template_id}                 | render with user parameters                                  |
| POST   | /api/templates/render-text                          | preview raw text                                             |
| GET    | /api/templates/analyze/{template_id}                | static analysis (dialog)                                     |
| GET    | /api/templates/{template_id}/include-candidates     | autocomplete targets                                         |
| POST   | /api/templates/copy-reference                       | build @@@{include ID} snippet                                |
| GET    | /api/templates/export                               | POO <sup>2</sup> L package export (poo2l_package 1.0)        |
| POST   | /api/templates/migrate-ids                          | sanitize unsafe ids on disk                                  |
| POST   | /api/templates/import                               | POO <sup>2</sup> L package import (check/execute ; 1.x only) |
| GET    | /api/templates/{template_id}/json                   | one-template package export                                  |
| GET    | /api/templates/container-export                     | container subtree package export                             |
| POST   | /api/templates/container-duplicate                  | deep-copy container                                          |
| POST   | /api/templates/import-single                        | import a one-template package (check/execute)                |
| GET    | /api/templates/extensions/namespaces                | foreign x_* namespaces + carrier counts                      |
| DELETE | /api/templates/{template_id}/extensions/{namespace} | remove one foreign x_* block                                 |
| POST   | /api/templates/extensions/purge                     | purge a namespace (or all) store-wide                        |

| Method | Path                                            | Purpose                                   |
|--------|-------------------------------------------------|-------------------------------------------|
| POST   | /api/templates/share-groups/{template_id}       | set group-sharing list (400 on refusal)   |
| GET    | /api/templates/sharing/{template_id}            | sharing info (second declaration is dead) |
| POST   | /api/templates/unshare-group/{template_id}      | ⚠ 500 on the default store                |
| POST   | /api/templates/transfer-ownership/{template_id} | ⚠ 500 on the default store                |
| GET    | /api/templates/user-templates                   | ⚠ legacy, 500 on the default store        |
| GET    | /api/templates/collections                      | container metadata rows                   |
| PUT    | /api/templates/collections/{path}               | create/update container                   |
| DELETE | /api/templates/collections/{path}               | delete container (members → trash)        |
| GET    | /api/templates/libraries                        | library manifests                         |
| PUT    | /api/templates/libraries/{name}                 | create/update library                     |
| DELETE | /api/templates/libraries/{name}                 | delete library (requires-guard)           |

## A.5 — Tags — /api/tags and /api/tags/v2 (chapter 5)

All hdr.

| Method | Path                         | Purpose                                    |
|--------|------------------------------|--------------------------------------------|
| GET    | /api/tags/list               | legacy list with counts                    |
| POST   | /api/tags/create             | create empty tag                           |
| POST   | /api/tags/rebuild-index      | rebuild the tag index                      |
| PUT    | /api/tags/color/{tag_name}   | set/clear color                            |
| PUT    | /api/tags/icon/{tag_name}    | set/clear icon                             |
| PUT    | /api/tags/acronym/{tag_name} | set/clear acronym ( $\leq 4$ , uppercased) |

| Method | Path                                            | Purpose                            |
|--------|-------------------------------------------------|------------------------------------|
| PUT    | /api/tags/short_name/{tag_name}                 | set/clear short name ( $\leq 15$ ) |
| PUT    | /api/tags/show_icon/{tag_name}                  | per-tag icon flag                  |
| GET    | /api/tags/{tag_name}                            | one tag (catch-all)                |
| PUT    | /api/tags/{tag_name}                            | rename (merge-capable)             |
| DELETE | /api/tags/{tag_name}                            | delete unused tag                  |
| GET    | /api/tags/v2/visible                            | visible v2 tags (bare array)       |
| POST   | /api/tags/v2                                    | create v2 tag (personal or group)  |
| PUT    | /api/tags/v2/{tag_id}                           | partial update (null clears)       |
| DELETE | /api/tags/v2/{tag_id}                           | delete (cascades)                  |
| POST   | /api/tags/v2/{tag_id}/important                 | per-user favourite flag            |
| POST   | /api/tags/v2/{tag_id}/share/{group_id}          | share (copy) into a group          |
| POST   | /api/tags/v2/{source_id}/merge-into/{target_id} | merge two tags                     |
| GET    | /api/tags/v2/prompt/{template_id}               | tag ids of a prompt                |
| POST   | /api/tags/v2/prompt/{template_id}/assign        | assign by name (smart)             |
| DELETE | /api/tags/v2/prompt/{template_id}/tag/{tag_id}  | unlink                             |

## A.6 — Presets, structure, builder, docs (chapter 5)

| Method | Path                                 | Auth | Purpose             |
|--------|--------------------------------------|------|---------------------|
| GET    | /api/templates/presets/{template_id} | hdr  | presets of a prompt |
| PUT    | /api/templates/presets/{template_id} | hdr  | replace presets     |
| DELETE | /api/templates/presets/{template_id} | hdr  | delete presets      |

| Method     | Path                                                                                         | Auth             | Purpose                                            |
|------------|----------------------------------------------------------------------------------------------|------------------|----------------------------------------------------|
| GET        | /api/templates/presets/                                                                      | hdr              | prompt ids with presets                            |
| GET        | /api/template-structure/config                                                               | —                | full creation-dialog catalogue                     |
| GET        | /api/template-structure/sections                                                             | —                | section definitions                                |
| GET        | /api/template-structure/standards                                                            | —                | standards + default                                |
| POST       | /api/template-structure/reload                                                               | —                | reload the YAML                                    |
| GET        | /api/template-builder/config                                                                 | hdr*             | user templates + sections (*falls back to default) |
| POST       | /api/template-builder/config                                                                 | hdr*             | save full config                                   |
| GET/POST   | /api/template-builder/templates                                                              | hdr*             | read / replace all templates                       |
| PUT/DELETE | /api/template-builder/templates/{key}                                                        | hdr*             | one template                                       |
| GET/POST   | /api/template-builder/sections                                                               | hdr*             | read / replace all sections                        |
| PUT/DELETE | /api/template-builder/sections/{key}                                                         | hdr*             | one section                                        |
| GET        | /api/template-builder/group-templates                                                        | hdr*             | all groups' templates                              |
| GET        | /api/template-builder/group/{group_id}/templates                                             | hdr* (member)    | one group's templates                              |
| POST       | /api/template-builder/group/{group_id}/templates/{key}                                       | hdr* (group adm) | share template                                     |
| DELETE     | /api/template-builder/group/{group_id}/templates/{key}                                       | hdr* (group adm) | remove shared template                             |
| GET        | /api/docs/guide/*, /api/docs/cgu, /api/docs/privacy, /api/docs/change-log, /api/docs/backend | —                | Markdown documentation content (17 fixed routes)   |

## A.7 — Groups and invitations (chapter 6)

| Method | Path                                      | Auth           | Purpose                        |
|--------|-------------------------------------------|----------------|--------------------------------|
| GET    | /api/groups                               | JWT            | all groups                     |
| POST   | /api/groups                               | JWT-adm        | create                         |
| GET    | /api/groups/me                            | JWT            | caller's groups                |
| GET    | /api/groups/me/admin                      | JWT            | groups the caller administers  |
| GET    | /api/groups/{group_id}                    | JWT            | details                        |
| PUT    | /api/groups/{group_id}                    | JWT-adm        | update                         |
| DELETE | /api/groups/{group_id}                    | JWT-adm        | delete (no members)            |
| GET    | /api/groups/{group_id}/members?scope=     | JWT            | member list (direct/effective) |
| POST   | /api/groups/{group_id}/members/{username} | JWT group/adm  | add member                     |
| DELETE | /api/groups/{group_id}/members/{username} | JWT group/adm  | remove member                  |
| POST   | /api/groups/{group_id}/admins/{username}  | JWT group/adm  | promote group admin            |
| DELETE | /api/groups/{group_id}/admins/{username}  | JWT group/adm  | demote group admin             |
| GET    | /api/groups/{group_id}/sub-groups         | JWT            | direct children                |
| GET    | /api/groups/{group_id}/tree               | JWT            | recursive subtree              |
| POST   | /api/groups/{parent}/sub-groups/{child}   | JWT group/adm  | attach child                   |
| DELETE | /api/groups/{parent}/sub-groups/{child}   | JWT group/adm  | detach child                   |
| GET    | /api/groups/user/{username}               | JWT (self/adm) | a user's groups                |
| PUT    | /api/groups/user/{username}               | JWT-adm        | replace memberships            |
| GET    | /api/groups/check-access                  | —              | ⚠ shadowed + broken            |

| Method | Path                              | Auth    | Purpose                                         |
|--------|-----------------------------------|---------|-------------------------------------------------|
| GET    | /api/invitations                  | JWT-adm | all invitations                                 |
| GET    | /api/invitations/pending          | JWT-adm | valid ones                                      |
| POST   | /api/invitations                  | JWT-adm | create (full token returned once)               |
| GET    | /api/invitations/validate/{token} | —       | validate                                        |
| DELETE | /api/invitations/{token}          | JWT-adm | revoke                                          |
| POST   | /api/invitations/register         | —       | register from invitation (⚠ 500 after creation) |
| POST   | /api/invitations/cleanup          | JWT-adm | purge old invitations                           |
| GET    | /api/invitations/by-email/{email} | JWT-adm | pending invitation for an email                 |
| POST   | /api/invitations/batch-import     | JWT-adm | CSV bulk invitations                            |

## A.8 — Administration — /api/admin (chapter 7)

| Method | Path                           | Auth    | Purpose                       |
|--------|--------------------------------|---------|-------------------------------|
| GET    | /api/admin/restart/status      | —       | pending restart state         |
| POST   | /api/admin/restart             | JWT-adm | restart (optional count-down) |
| POST   | /api/admin/restart/cancel      | JWT-adm | cancel scheduled restart      |
| GET    | /api/admin/migration/storages  | JWT-adm | storage catalogue             |
| POST   | /api/admin/migration/config    | JWT-adm | write [database] config       |
| GET    | /api/admin/migration/config    | JWT-adm | current config (masked)       |
| GET    | /api/admin/attachments/backend | JWT-adm | attachment store              |

| Method | Path                           | Auth      | Purpose                 |
|--------|--------------------------------|-----------|-------------------------|
| PUT    | /api/admin/attachments/backend | JWT-admin | switch attachment store |

## A.9 — Versioning, git, diff (chapter 8)

| Method | Path                                  | Auth      | Purpose                             |
|--------|---------------------------------------|-----------|-------------------------------------|
| GET    | /api/versions/{template_id}           | hdr       | version list                        |
| GET    | /api/versions/{template_id}/{version} | hdr       | prompt at a version                 |
| POST   | /api/versions/{template_id}/migrate   | hdr       | git history → versions (overwrites) |
| POST   | /api/versions/{template_id}/copy      | hdr       | ⚠ reserves id only, no write        |
| POST   | /api/versions/migrate-all             | hdr       | migrate every prompt                |
| GET    | /api/git/status                       | —         | repo status ( applicable gate)      |
| POST   | /api/git/init                         | —         | initialize git                      |
| POST   | /api/git/remote                       | —         | add/replace remote                  |
| DELETE | /api/git/remote/{name}                | —         | remove remote                       |
| POST   | /api/git/commit                       | —         | manual commit (+auto-push)          |
| POST   | /api/git/push                         | —         | ⚠ response meaningless              |
| POST   | /api/git/pull                         | —         | ⚠ response meaningless              |
| GET    | /api/git/log                          | —         | ⚠ broken                            |
| GET    | /api/git/diff/{commit_hash}           | —         | ⚠ broken                            |
| GET    | /api/git/remotes                      | —         | ⚠ broken                            |
| GET    | /api/git/config                       | —         | [git] settings                      |
| PUT    | /api/git/config                       | JWT-admin | update [git] settings               |

| Method | Path              | Auth | Purpose            |
|--------|-------------------|------|--------------------|
| POST   | /api/diff/compare | —    | semantic text diff |

## A.10 — Scoring and migration (chapter 9)

| Method | Path                                     | Auth        | Purpose                                         |
|--------|------------------------------------------|-------------|-------------------------------------------------|
| GET    | /api/scoring/goals                       | —           | scoring goals                                   |
| POST   | /api/scoring/score                       | —           | score a text                                    |
| POST   | /api/scoring/sections                    | —           | detect sections                                 |
| POST   | /api/scoring/regenerate                  | —           | rebuild with section toggles ( text in query !) |
| GET    | /api/scoring/section-lengths             | —           | length configuration                            |
| GET    | /api/admin/migration/status              | hdr/JWT-adm | backend + exports (⚠ 500 on default backend)    |
| POST   | /api/admin/migration/export              | hdr/JWT-adm | export the store                                |
| GET    | /api/admin/migration/download/{filename} | hdr/JWT-adm | download export                                 |
| POST   | /api/admin/migration/import              | hdr/JWT-adm | import (multipart)                              |
| POST   | /api/admin/migration/validate            | hdr/JWT-adm | dry-run validation                              |
| DELETE | /api/admin/migration/exports/{filename}  | hdr/JWT-adm | delete export                                   |
| GET    | /api/admin/migration/backends            | hdr/JWT-adm | backend catalogue                               |

## A.11 — Events and status (chapter 10)

| Method | Path               | Auth | Purpose                               |
|--------|--------------------|------|---------------------------------------|
| GET    | /api/events/stream | —    | SSE stream (identity in query params) |

| Method | Path                        | Auth                | Purpose                         |
|--------|-----------------------------|---------------------|---------------------------------|
| GET    | /api/events/status          | —                   | connected count + last_shutdown |
| POST   | /api/events/notify/shutdown | hdr (no admin gate) | broadcast shutdown              |
| POST   | /api/events/notify/ready    | —                   | broadcast ready                 |
| GET    | /api/status                 | —                   | liveness                        |
| GET    | /api/queue/stats            | —                   | △ hard-coded stub               |
| GET    | /api/routes                 | —                   | route introspection (JSON)      |
| GET    | /api/routefile              | —                   | route introspection (HTML)      |

## A.12 — Analytics, alerts, diagnostics (chapter 11)

| Method | Path                                      | Auth                 | Purpose            |
|--------|-------------------------------------------|----------------------|--------------------|
| POST   | /api/analytics/track                      | hdr                  | record event       |
| GET    | /api/analytics/summary                    | hdr                  | rolling summary    |
| GET    | /api/analytics/templates/{template_id}    | hdr                  | per-template stats |
| GET    | /api/analytics/export/csv                 | hdr                  | raw CSV export     |
| GET    | /api/analytics/export/json                | hdr                  | raw JSON export    |
| GET    | /api/analytics/dashboard/system           | hdr-adm              | system dashboard   |
| GET    | /api/analytics/dashboard/groups           | hdr-adm              | all groups summary |
| GET    | /api/analytics/dashboard/group/{group_id} | hdr-adm or group adm | group dashboard    |
| GET    | /api/analytics/dashboard/user/{username}  | hdr (self/adm)       | user dashboard     |
| GET    | /api/analytics/dashboard/me               | hdr                  | own dashboard      |

| Method | Path                                             | Auth                 | Purpose                     |
|--------|--------------------------------------------------|----------------------|-----------------------------|
| GET    | /api/analytics/export/dashboard/system           | hdr-adm              | report download             |
| GET    | /api/analytics/export/dashboard/group/{group_id} | hdr-adm or group adm | report download             |
| GET    | /api/admin/alerts                                | hdr-adm              | paginated alerts            |
| GET    | /api/admin/alerts/recent                         | hdr-adm              | unacknowledged feed         |
| POST   | /api/admin/alerts/{alert_id}/acknowledge         | hdr-adm              | acknowledge one             |
| POST   | /api/admin/alerts/acknowledge-all                | hdr-adm              | acknowledge all             |
| GET    | /api/admin/alerts/config                         | hdr-adm              | alerting config             |
| GET    | /api/admin/alerts/stats                          | hdr-adm              | counters                    |
| POST   | /api/admin/alerts/test                           | hdr-adm              | fire a test alert           |
| GET    | /api/dev/diagnostic                              | JWT-adm              | full diagnostic (secrets !) |
| GET    | /api/dev/diagnostic/user                         | JWT                  | sanitized diagnostic        |
| GET    | /api/dev/diagnostic/report                       | JWT-adm              | text report                 |

### A.13 — Config, tasks, license (chapter 12)

| Method | Path                         | Auth | Purpose                                 |
|--------|------------------------------|------|-----------------------------------------|
| GET    | /api/config                  | —    | config snapshot + live_reload / boot_id |
| POST   | /api/config                  | —    | partial config update                   |
| GET    | /api/config/exclude-patterns | —    | exclusion patterns                      |
| POST   | /api/config/exclude-patterns | —    | replace patterns                        |
| POST   | /api/config/reset            | —    | reset to defaults                       |

| Method | Path                               | Auth      | Purpose                                                              |
|--------|------------------------------------|-----------|----------------------------------------------------------------------|
| —      | /api/tasks/*                       | —         | ⚠ router not mounted (desktop-era queue) — every path is a plain 404 |
| GET    | /api/license/status                | hdr       | plan and limits                                                      |
| POST   | /api/license/activate              | hdr-adm   | activate a key                                                       |
| POST   | /api/license/deactivate            | hdr-adm   | reset to freemium                                                    |
| GET    | /api/license/portal                | hdr-adm   | customer portal URL                                                  |
| GET    | /api/license/checkout/{variant_id} | hdr-adm   | checkout URL                                                         |
| POST   | /api/webhooks/lemons-queezy        | signature | provider webhook                                                     |
| GET    | /api/webhooks/lemons-queezy/test   | —         | reachability probe                                                   |

## A.14 — The Lab — /api/tests (appendix D)

Mounted only with `--lab` ; all routes loopback-only.

| Method | Path                                  | Purpose                       |
|--------|---------------------------------------|-------------------------------|
| GET    | /api/tests/status                     | availability probe            |
| GET    | /api/tests/batches                    | batch inventory               |
| GET    | /api/tests/                           | fixture list (trailing slash) |
| GET    | /api/tests/{name}                     | fixture spec                  |
| POST   | /api/tests/{name}/run                 | run one                       |
| POST   | /api/tests/run-all                    | run all                       |
| POST   | /api/tests/batches/{batch}/run        | run a batch                   |
| POST   | /api/tests/run-all/stream             | NDJSON stream                 |
| POST   | /api/tests/batches/{batch}/run/stream | NDJSON stream                 |

A live server can always be asked for the authoritative list: `GET /api/routes` .

## Appendix B — Data Model Reference

The recurring JSON objects of the API, collected in one place. Each model notes the chapter where its behavior is discussed.

### B.1 — Template (chapter 3)

The stored prompt, as serialized by the API ( `name` , `created_at` , `updated_at` duplicate `title` , `created` , `modified` ):

```
{
 "id": "3f2a9c1e-uuid",
 "template_id": "CODE_REVIEW",
 "title": "Code review",
 "name": "Code review",
 "content": "@@@name Code review\n@@@param lang...\n...",
 "tags": ["dev"],
 "created": "2026-01-02T10:00:00", "created_at": "2026-01-02T10:00:00",
 "modified": "2026-06-01T09:30:00", "updated_at": "2026-06-01T09:30:00",
 "description": "...",
 "notes": "",
 "color": "#ff8800",
 "locked": false,
 "important": false,
 "disabled": false,
 "is_example": false,
 "deleted": false,
 "deleted_at": null,
 "author": "jacques",
 "short_name": "CR",
 "shared_with_groups": [],
 "order": 10,
 "references": [],
 "scope": "shared", "scope_id": "", "owner": "", "visibility": "personal",
 "wip": false,
 "no_scoring": false,
 "attachments": [],
 "structure_policy": {},
 "parameters": { "lang": { "type": "text", "label": "lang", "default": "fr", "required": true },
 "constants": {},
 "inherits": null,
 "aliases": [],
 "deprecated": null,
 "collection": "Dev/Reviews",
 "virtual": false,
 "source": {}
}
```

```
"extensions": {}
}
```

Key invariants: `id` is the stable storage UUID; `template_id` is the logical POO<sup>2</sup>L identifier (canonical UPPERCASE, unique per collection path only); `tags[0]` is the primary tag; `content` carries the inline declarations and the model-level `attachments` list stays empty on save. `extensions` holds foreign `x_*` blocks from imported POO<sup>2</sup>L packages verbatim (unversioned; `x_izibuddy` is never stored there — it is generated from the native fields at export time). SQL backends persist it in a dedicated `extensions` JSON column, MongoDB as a field; those backends otherwise remain lossy for fields outside their columns — the new column only restores the foreign-block round-trip.

## B.2 — Attachment (chapters 3-4)

```
{ "id": "a1", "type": "included", "name": "SPEC",
 "filename": "spec.pdf", "mime_type": "application/pdf", "size": 12345, "data": "<ba
```

```
{ "id": "a2", "type": "to_include", "name": null,
 "description": "The contract PDF", "accepted_types": ["pdf"], "required": true }
```

`data` is always base64 in API responses (storage references are re-materialized server-side). `name` is the POO<sup>2</sup>L handle used by `@@@attachment` references.

## B.3 — Parameter definition (chapter 4)

Produced by analysis and stored in `parameters` :

```
{ "type": "select", "label": "Langue", "default": "fr", "required": true, "options":
```

Always `type`, `label`, `default`, `required`; extras by type: `min/max` (number), `options` (select), `rows` (textarea), `minlen/maxlen` (text/textarea), `pattern` (text), `section` (conditional display).

## B.4 — Render warning (chapter 4)

```
{ "type": "missing_description", "message": "MISSING_DESCRIPTION: ...", "blocking": fa
```

`type` is the lowercase engine diagnostic code; `blocking` and `handle` are optional. Hard-error `error_type` values: `unimplemented_virtual_section`, `override_of_final_section`, `override_of_final_template`, `include_of_virtual_section`, `ambiguous_include`, `system_variable_override`, `constraint_violation`, `select_invalid_option`, plus the app-level `disabled_template`, `disabled_reference`, `circular_include`, `diamond_include`, `not_found`.

## B.5 — Collection metadata and library manifest (chapter 3)

```
{ "kind": "collection", "description": "...", "color": "#123456",
 "label": "Revues", "tags": [], "capabilities": [], "inherits": "Base" }
```

`kind` ∈ `folder` | `collection` | `library` | `project`. Library manifest:

```
{ "description": "...", "color": null, "label": null,
 "requires": ["CORE"], "collections": [], "templates": [], "folder_path": "" }
```

## B.6 — User (chapter 2)

```
{
 "username": "jean.dupont",
 "email": "jean@example.com",
 "is_admin": false,
 "is_superadmin": false,
 "is_active": true,
 "groups": ["dev"],
 "status": "active",
 "email_verified": true,
 "has_2fa": false,
 "created_at": "2026-01-13T14:32:00.123456",
 "last_login": "2026-07-12T09:15:00.000001"
}
```

## B.7 — Web settings (chapter 6)

The merged `web.json` document (server defaults + stored values):

```
{
 "lang": "en",
 "theme": "light",
 "lastAccess": "2026-07-12T09:00:00",
 "ui_level": "simple",
 "active_preset": "standard",
 "features": { "filterImportant": true },
 "feature_configs": {},
 "active_config": null,
 "selection_mode": "simple",
 "default_no_scoring": false,
 "tags_as_icons": false,
 "hide_tag_folders": false,
 "container_colors": { "folder": { "light": "#ffffff", "dark": "#0a0a0a" },
 "collection": { "light": "#d1fae5", "dark": "#0f3320" } },
 "themes": {},
}
```

```
"active_theme": null,
"table_config": {}
}
```

## B.8 — Group (chapter 6)

```
{
 "id": "dev",
 "name": "Développeurs",
 "description": "",
 "admins": ["alice"],
 "members": ["bob", "carol"],
 "subgroups": ["dev-front"],
 "created_at": "2026-01-10T08:00:00",
 "created_by": "alice"
}
```

## B.9 — Invitation (chapter 6)

```
{
 "token": "aB3xYz9q...",
 "email": "new@company.com",
 "groups": ["dev"],
 "invited_by": "alice",
 "created_at": "2026-07-01T10:00:00",
 "expires_at": "2026-07-08T10:00:00",
 "status": "pending",
 "accepted_at": null,
 "is_valid": true
}
```

`token` is truncated everywhere except the creation response. `status` ∈ `pending` | `accepted` | `revoked` (+ computed `expired`).

## B.10 — Tag, v2 (chapter 5)

```
{
 "id": "8f14e45f-ceed-467f-a8f0-1c6b1f0a2d31",
 "name": "python",
 "owner_user": "jean.dupont",
 "owner_group": null,
 "description": null,
 "color": "#3fb950",
 "icon": "🐍",
 "acronym": "PY",
 "short_name": null,
}
```

```

"show_icon": false,
"synonyms": ["py"],
"created_at": "2026-07-12T09:15:00.123456",
"created_by": "jean.dupont",
"important_for_me": true
}

```

Exactly one of `owner_user` / `owner_group` is set. `important_for_me` appears only in `/visible`. The legacy tag shape (chapter 5) is `{name, count, color, icon, acronym, short_name, show_icon}`.

## B.11 — Version entry (chapter 8)

```

{
 "format": 1,
 "version": "1.3",
 "sha": "a1b2c3d4",
 "datetime": "2026-07-01 14:32:00",
 "operation": "u",
 "note": "Refonte de la section task",
 "checksum": "e3b0c442"
}

```

`operation` ∈ `c` (create), `u` (update), `r` (restore), `m` (migrated).

## B.12 — Diff block (chapter 8)

```

{
 "id": "block_1",
 "type": "paragraph",
 "status": "modified",
 "content_old": "Cuire 10 minutes.",
 "content_new": "Cuire 12 minutes.",
 "old_spans": [{ "text": "10", "changed": true }],
 "new_spans": [{ "text": "12", "changed": true }]
}

```

`type` ∈ `title` | `code` | `list` | `paragraph` ; `status` ∈ `unchanged` | `modified` | `added` | `removed` ; spans only in `word_mode` on modified blocks.

## B.13 — Score result (chapter 9)

Core fields: `score`, `score_min`, `score_max`, `score_range`, `base_fraction`, `coherence_fraction`, `badge`, `badge_tooltip`, `goal`, `has_forced_error`, `missing_required`, `active_sections`, `present_sections`, `missing_sections`, `optio-`

nal\_sections, coherence\_checks (each {check\_id, label, mode, sections, weight, raw\_score, effective\_weight, contribution, missing\_sections}), failed\_checks, dynamic\_tooltip, coverage.

## B.14 — Alert (chapter 11)

```
{
 "id": "3f8c1e2a-uuid",
 "alert_type": "failed_login",
 "severity": "warning",
 "message": "Multiple failed login attempts for user bob: 5 attempts",
 "data": { "user": "bob" },
 "created_at": "2026-07-12T07:55:02.412345Z",
 "acknowledged": false,
 "acknowledged_by": null,
 "acknowledged_at": null
}
```

alert\_type ∈ error\_rate | failed\_login | brute\_force | slow\_requests | disk\_space | user\_locked | security\_event | system\_error | custom ; severity ∈ info | warning | critical.

## B.15 — SSE event (chapter 10)

```
{ "type": "shutdown", "timestamp": "2026-07-12T09:41:00.123456", "initiated_by": "admin" }
```

type ∈ connected | shutdown | restart | ready ; timestamp absent on connected, initiated\_by only on shutdown / restart.

## B.16 — License status (chapter 12)

```
{
 "plan": "pro",
 "plan_name": "Pro",
 "type": "subscription",
 "template_limit": null,
 "template_count": 210,
 "can_create": true,
 "status": "active",
 "expires_at": "2026-12-01T00:00:00.000000Z",
 "customer_email": "owner@company.com",
 "product_name": "IziBuddy Pro"
}
```

plan ∈ freemium | pro ; type ∈ none | one\_time | subscription ; status ∈ active | cancelled | expired | inactive.

## B.17 — Task (chapter 12)

```
{ "id": "task_a1b2c3d4_1720795200123", "type": "concat", "project": "my-project", "dura
```

Queue stats: {pending, active, workers, processed, errors, running, up-time}.

## Appendix C — Error Handling Reference

Everything a frontend's HTTP layer needs in one place: which envelopes exist, which status codes mean what, how domain errors are encoded, and the catalogue of known defects to code defensively around.

### C.1 — The two error envelopes

**HTTP-status errors** use FastAPI's standard envelope:

```
{ "detail": "Template not found" }
```

`detail` is usually a string; Pydantic validation failures (422) carry a list instead:

```
{ "detail": [{ "loc": ["body", "name"], "msg": "Field required", "type": "missing" }] }
```

**Domain failures inside HTTP 200** are the second idiom, dominant on the templates router and the git router:

```
{ "success": false, "error": "...", "error_type": "disabled_template", "handle": "..." }
```

`error_type` and `handle` are optional and power localized error UIs. The practical rule: **branch on success when the field is present, and treat the HTTP status as transport-level only.**

### C.2 — Status codes in practice

| Status | Meaning in this API                                                                                                                                                            |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 200    | success — <i>or</i> a domain failure with <code>success: false</code>                                                                                                          |
| 204    | PNA preflight response                                                                                                                                                         |
| 307    | trailing-slash redirect (e.g. <code>/api/tests</code> → <code>/api/tests/</code> )                                                                                             |
| 400    | malformed input or refused operation (group sharing, tag constraints, group management, license key format)                                                                    |
| 401    | "Not authenticated" / "Invalid or expired token" (JWT routes); "X-IziBuddy-User header required" (header routes); "Invalid credentials" (login); "Invalid signature" (webhook) |
| 403    | "Account is disabled", "Admin access required", "System admin access required", group-admin refusals, "Lab is loopback-only"                                                   |

| Status | Meaning in this API                                                                                                                                                                                                 |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 404    | unknown resource — but note the templates router mostly reports not-found as 200/ <code>success: false</code> ; the whole <code>/api/tasks/*</code> family 404s because that router is not mounted (chapter 12)     |
| 409    | git routes on a non-filesystem storage backend                                                                                                                                                                      |
| 422    | Pydantic validation ; also a <i>missing required header parameter</i> (e.g. <code>/api/events/notify/shutdown</code> without the identity header)                                                                   |
| 500    | unexpected exception ( <code>detail</code> is the exception text)                                                                                                                                                   |
| 501    | dashboard export format whose optional dependency is missing                                                                                                                                                        |
| 503    | subsystem not initialized ( <code>"Config manager not initialized"</code> , <code>"Analytics not initialized"</code> , <code>"Alert manager not initialized"</code> ) — normally transient, retry later during boot |

### C.3 — Authentication failures

The complete 401/403 matrix is in chapter 2. The frontend recipe : a global handler that (1) retries 503s with backoff during boot, (2) routes JWT 401s to the login screen, (3) surfaces 403s as permission errors without logging the user out.

### C.4 — Render and engine errors

Rendering failures are HTTP 200 with `success: false` and an `error_type` (chapter 4): the engine's own diagnostic codes in lowercase (`ambiguous_include`, `constraint_violation`, `system_variable_override`, ...) plus the app-level types (`disabled_template`, `disabled_reference`, `circular_include` — with `cycle_path` —, `diamond_include`, `not_found`). The optional `handle` names the offending include or collection. Save-time guards reuse the same vocabulary (`SYSTEM_VARIABLE_OVERRIDE` in the `error` string, `error_type: "system_variable_override"`).

Container and library mutations return typed error **codes** in the `error` field: `INHERITS_DESCENDANTS_BLOCK_DELETE`, `REQUIRES_DESCENDANTS_BLOCK_DELETE`, `INVALID_LIBRARY_NAME`, `CIRCULAR_LIBRARY_DEPENDENCY`, `UNDEFINED_REFERENCE`: ... — stable strings a frontend can match on.

### C.5 — Message localization

Human-readable `message` and `detail` strings are a mixed bag : some are localized to the **server's** language (backend i18n), some are hardcoded English, some hardcoded French (scoring tooltips, parts of the self-registration flow). Never parse them ; match on status codes, `success` booleans, `error_type` and the typed error codes above, and generate your own display strings.

## C.6 — Non-blocking warnings

Success responses may carry a `warnings` array (chapters 3–4). Two warning types deserve dedicated handling: `git_push_error` (the data is saved locally, only the push failed — retry later or surface a sync indicator) and `invalid_registry_entry` (another prompt in the store is unparsable and was excluded from resolution — worth a maintenance hint).

## C.7 — Known defects catalogue

Verified against the source at the time of writing; each is flagged in its chapter. Defensive frontends should assume they may be fixed at any time — code for the *documented intended* behavior and tolerate the defective one.

| Where                                                                                                                                                              | Defect                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <code>GET /api/users/admins</code>                                                                                                                                 | shadowed by <code>GET /api/users/{username}</code> — unreachable; use <code>GET /api/auth/users</code>                        |
| <code>GET /api/groups/check-access</code>                                                                                                                          | shadowed by <code>GET /api/groups/{group_id}</code> and its backing methods don't exist                                       |
| <code>POST /api/invitations/register</code>                                                                                                                        | 500 <i>after</i> account creation, before invitation consumption and group assignment — treat as « account probably created » |
| <code>POST /api/templates/unshare-group/{id}</code> , <code>POST /api/templates/transfer-ownership/{id}</code> ,<br><code>GET /api/templates/user-templates</code> | 500 on the default store (manager surface missing)                                                                            |
| <code>POST /api/templates/share\ unshare\ duplicate/{id}</code>                                                                                                    | always <code>not_supported</code> on the default store                                                                        |
| second <code>GET /api/templates/sharing/{template_id}</code>                                                                                                       | dead route (first declaration wins)                                                                                           |
| <code>POST /api/templates/render/{id}</code> , <code>/render-text</code>                                                                                           | <code>section_states</code> accepted but ignored                                                                              |
| <code>POST /api/git/push</code> , <code>POST /api/git/pull</code>                                                                                                  | operation runs, response meaningless                                                                                          |

| Where                                                      | Defect                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET /api/git/log , /api/git/diff/{hash} , /api/git/remotes | always fail (missing/incompatible manager methods)                                                                                                                                                                                                                                                               |
| GET / POST / PUT /api/templates/hidden...                  | 500-style domain failure for grouped users : builds group ids with <code>g.get('id')</code> over a list of id strings, so <code>200 {"success": false, "error": "'str' object has no attribute 'get'"}</code> whenever the caller belongs to $\geq 1$ group ; correct for users in no group. /list is unaffected |
| POST /api/versions/{id}/copy                               | reserves the id but does not write the copy                                                                                                                                                                                                                                                                      |
| POST /api/versions/{id}/migrate , /migrate-all             | overwrite existing version lists                                                                                                                                                                                                                                                                                 |
| POST /api/scoring/score                                    | <code>weight_overrides</code> accepted but ignored ; unknown <code>goal</code> silently falls back                                                                                                                                                                                                               |
| POST /api/scoring/regenerate                               | <code>text</code> binds as a query parameter                                                                                                                                                                                                                                                                     |
| POST /api/config                                           | <code>check_status_interval</code> / <code>queue_stats_interval</code> accepted but never applied                                                                                                                                                                                                                |
| GET /api/queue/stats                                       | hard-coded stub                                                                                                                                                                                                                                                                                                  |
| POST /api/events/notify/shutdown                           | no admin gate ; missing header yields 422                                                                                                                                                                                                                                                                        |
| GET /api/analytics/export/csv                              | naive CSV (no quoting), not admin-gated                                                                                                                                                                                                                                                                          |
| POST /api/license/deactivate                               | local-only ; provider seat not released                                                                                                                                                                                                                                                                          |

## C.8 — Security-posture summary

The `X-IziBuddy-User` header is an **assertion**, not a credential (chapter 2) ; several routers trust it, and some routes are anonymous by omission (user settings by path, git plumbing, scoring, config). The JWT layer genuinely protects `/api/templates` , `/api/auth` , `/api/groups` , `/api/invitations` , `/api/admin` and `/api/dev` . For any deployment beyond a trusted machine or network, terminate real authentication at a reverse proxy and forward both identity headers from there.

## Appendix D — The Lab : Test and Diagnostic Routes

The **IziBuddy Lab** is an embedded test harness : fixtures covering the rendering pipeline, the conformance pack of the POO<sup>2</sup>L standard, scoring and collection semantics, runnable from the browser. Its router ( `/api/tests` ) exists only under explicit opt-in and is loopback-guarded. A frontend that integrates the Lab gets a self-testing installation ; one that ignores it loses nothing functional.

### D.1 — Availability rules

- Mounted **only** when the server starts with `--lab` or the environment variable `IZI-BUDDY_LAB` set to `1 / true / yes / on` . The flag is read once at startup ; without it every `/api/tests/*` path is a plain 404 and nothing can enable it at runtime.
- Even when mounted, every route rejects non-loopback clients with `403 {"detail": "Lab is loopback-only"}` .
- The UI contract: probe `GET /api/tests/status` at boot — `200` shows the Lab menu, `404` hides it.
- In `--lab` mode the process runs under a supervisor that watches the sources and re-spawns the worker on modification (5 s debounce). The worker broadcasts the SSE `shutdown / ready` events (chapter 10) around each respawn, and `live_reload` flips to `true` in `GET /api/config` .

### D.2 — The fixture corpus

Fixtures form one flat namespace. Each carries `title` , `category` , a `batch` tag and a runner discriminator. Two families :

**App fixtures** — pin application semantics :

| Batch                             | Runner                | What it pins                                                                                                                 |
|-----------------------------------|-----------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>front-tend_subst</code>     | <code>frontend</code> | the browser-side parameter substitution step (exact-label <code>@@@=name</code> , default fallback, literal undeclared refs) |
| <code>poole-vel_detec-tion</code> | <code>frontend</code> | the L0-L5 POO <sup>2</sup> L complexity classification used by the editor                                                    |
| <code>scoring</code>              | <code>scoring</code>  | scoring-engine outcomes (badges, forced errors, optional sections)                                                           |

| Batch               | Runner                                | What it pins                                                                                                                        |
|---------------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| collec-<br>tions_oo | render +<br>collec-<br>tion_validator | OO <sup>2</sup> collection semantics: inheritance chains, virtual-template warnings, parent resolution, save-time validator rejects |

**Conformance pack** — every case of the POO<sup>2</sup>L standard-conformance test suite becomes a fixture `conf:<block>/<level>/<case>` with `batch: "conformance:<block>:<level>"`, across blocks `simple`, `combination`, `complexity` and `stress`. The pack is the grammar authority; each case is a `source.pool`, an optional registry of `.pool` files and context, and either an expected text or an expected typed engine error.

Render fixtures execute against an **ephemeral in-memory prompt store** through the exact production render path — the user's data is never touched. `frontend-runner` fixtures cannot run server-side: the server returns them as `{"passed": null, "skipped": true, "frontend": {fn, input}}` and the browser Lab executes the named JS function locally, patching the report.

## D.3 — Routes

All under `/api/tests`, all loopback-only:

| Method & path                                                                        | Effect                                                                                                                                                                              |
|--------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GET <code>/status</code>                                                             | availability probe: <code>{"enabled": true, "fixture_count": 440}</code>                                                                                                            |
| GET <code>/batches</code>                                                            | batch inventory: <code>{"batches": [{"name": "scoring", "count": 4}, {"name": "conformance:simple:L1", "count": 24}, ...]}</code>                                                   |
| GET <code>/</code>                                                                   | every fixture ( <code>name</code> , <code>title</code> , <code>category</code> , <code>batch</code> ) — note the <b>trailing slash</b> ; without it FastAPI answers a 307 re-direct |
| GET <code>/ {name}</code>                                                            | the raw fixture spec; 404 <code>{"detail": "Unknown fixture: ..."}</code>                                                                                                           |
| POST <code>/ {name}/run</code>                                                       | run one fixture, returns the result object below                                                                                                                                    |
| POST <code>/run-all</code>                                                           | run everything; aggregate report                                                                                                                                                    |
| POST <code>/batches/{batch}/run</code>                                               | one batch; 404 <code>{"detail": "Unknown batch: ..."}</code>                                                                                                                        |
| POST <code>/run-all/stream</code> ,<br>POST <code>/batches/{batch}/run/stream</code> | NDJSON streaming variants                                                                                                                                                           |

## D.4 — Result shapes

A single run returns (superset — keys vary by runner):

```
{
 "passed": true,
 "matched": true,
 "expects_fail": false,
 "actual": "rendered text",
 "expected": "rendered text",
 "warnings": [{ "type": "unimplemented_virtual_template", "message": "..." }],
 "diff": "",
 "error": null,
 "inputs": { "templates": [], "render": "id", "params": {}, "system_vars": {} },
 "name": "coll_virtual_unmet_warns_at_render",
 "title": "Virtual prompt declared but never overridden → warning surfaces",
 "category": "Collections 00²",
 "batch": "collections_oo"
}
```

Runner notes: `expects_fail: true` marks canary fixtures whose text diff *must* fail; scoring fixtures return dict-valued `actual/expected` plus a `rules` block and a per-check `coherence_breakdown` so the UI can explain the score; validator fixtures compare error-code strings; conformance fixtures compare text (trailing newlines ignored) or a typed `"<error CODE>"`.

The aggregate report:

```
{
 "total": 440,
 "passed": 412,
 "failed": 0,
 "skipped": 28,
 "results": { "<fixture-name>": { ...result object... } }
}
```

`skipped` counts the frontend-runner fixtures; a browser Lab re-runs them locally and patches the totals.

The streaming variants emit one JSON object per line ( `Content-Type: application/x-ndjson`, buffering disabled):

```
{"kind": "result", "index": 1, "total": 440, "name": "fe_subst_exact_lowercase", "re...
{"kind": "done", "total": 440, "passed": 412, "failed": 0, "skipped": 28}
```

The batch variant validates the batch name **before** streaming starts, so a 404 is still an ordinary JSON error.

## D.5 — Integrating the Lab in a custom frontend

1. Probe `GET /api/tests/status` ; hide everything on 404.
2. Build the tree from `GET /batches` + `GET /` .
3. Run with the streaming endpoints to drive a progress bar; fall back to the synchronous ones for simplicity.
4. If you want the frontend-runner fixtures to count, implement the three referenced JS functions (parameter substitution, level computation, gated-range detection) or ship them from the bundled web app — the server tells you which function and input each skipped fixture expects.

# Resources

---

**The backend** lives in the `src/` directory of the IziBuddy repository :

- `src/main.py` — application factory, middleware stack, router registration ;
- `src/api/` — one module per API domain (templates, users, tags, events, ...);
- `src/core/` — business managers the routers delegate to.

**Live API documentation.** A running server also exposes the machine-generated OpenAPI documentation at `/docs` (Swagger UI) and `/redoc` (ReDoc). This book is the narrative companion : conventions, flows and semantics the generated pages cannot express.

**Companion guides.** Template rendering is delegated to the POO<sup>2</sup>L engine ; its programming interface is the subject of the *POO<sup>2</sup>L Engine API Guide* (book 08), and the language itself of the *POO<sup>2</sup>L Language Reference* (book 02).

## Downloads

- PDF : 09-backend-fastapi\_v0.9\_en\_20261008.pdf
- EPUB : 09-backend-fastapi\_v0.9\_en\_20261008.epub
- Markdown : 09-backend-fastapi\_v0.9\_en\_20261008.md