Sunesix Marketplace: App Architecture & Developer Guide

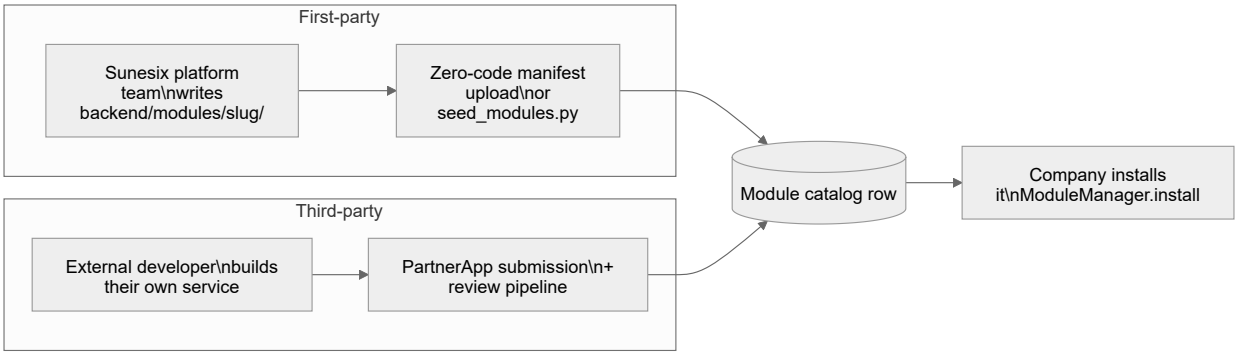
August 4, 2026

CONTENTS

Sunesix Marketplace: App Architecture & Developer Guide

1. What the marketplace is

The Sunesix marketplace is the catalog of everything a company can install onto its tenant: first-party business modules built and deployed by the Sunesix platform team, and third-party apps built and hosted by outside vendors. Both end up as the same kind of catalog row — a `Module` — and both go through the same install/enable/disable/RBAC machinery once listed. What differs entirely is how each one gets *onto* the platform, and that's the one distinction this document exists to make clear before anything else.



First-party modules are Django apps that live in this codebase, under `backend/modules/<slug>/`, deployed alongside the platform itself and reviewed by the Sunesix platform team before they ship. Their code runs **in-process**, inside the same Django deployment as everything else.

Third-party apps (`partner_apps`) are the opposite in one specific, load-bearing way: **no third-party code ever executes on Sunesix infrastructure.** A vendor's app is a service the vendor hosts and runs themselves. Sunesix either calls out to it (a guardrail vendor, invoked mid-request for an allow/block verdict) or the vendor calls in through a narrow, capability-scoped API (an AI-agent builder, reading exactly the data a Company Admin consented to). This isn't a limitation waiting to be lifted — it's the reason a

careless or malicious third-party integration can't compromise every tenant at once. A shared, multi-tenant monolith with no per-app process sandboxing means the only safe boundary is the network, not a runtime plugin host.

This guide's primary audience is the third-party path — if you're an external developer or vendor building something to list in the marketplace, section 3 is your actual reference. Section 2 covers both formats for completeness, since understanding what a first-party module's manifest looks like also clarifies what a `Module` catalog row actually *is* underneath a third-party listing.

---

2. App formats

2.1 FIRST-PARTY MODULE MANIFEST (`MANIFEST.JSON`)

A Sunesix Superadmin can publish or update a catalog entry by uploading a small `.zip` from the Admin Panel instead of hand-editing `seed_modules.py`. The zip may **only ever contain data** — a manifest file plus, optionally, one icon image. It can never contain Python, JavaScript, shell scripts, or any other executable content; the upload pipeline actively rejects anything that looks like code and never writes the archive's contents to disk or imports/executes anything from it.

This is deliberate. The catalog-metadata half of publishing an app is upload-driven; the code half still ships through a normal platform deploy, exactly like every module in `backend/modules/` already does. Uploading a manifest for a slug with no corresponding deployed Django app creates an **unpublished** listing (`is_active=false`) and says so in the response — it never pretends an app is usable when nothing implements it.

Archive layout:

```
manifest.json      # required, exactly this name, at the archive root
icon.png|.svg|.jpg  # optional, at most one
```

No subdirectories. No other files. Archives over 2MB or with more than 10 entries are rejected outright (zip-bomb/junk mitigation), and every entry name is checked for path traversal and blocked code-like extensions (`.py`, `.js`, `.sh`, `.exe`, `.dll`, …) or the

Unix executable bit, regardless of extension.

Manifest fields:

Field	Type	Required	Notes
`slug`	string	yes	Lowercase letters/numbers/underscores. Must match an already-deployed `modules/<slug>/` app to be installable.
`name`	string	yes	Display name.
`description`	string	no	
`version`	string	no	Semver, default `1.0.0`.
`category`	string	no	
`icon`	string	no	A lucide icon identifier — not the optional icon asset in the zip.
`tags`	string[]	no	Marketplace search keywords.
`dependencies`	string[]	no	Other module slugs required first.
`min_platform_version`	string	no	
`changelog`	object[]	no	`{version, date, summary, breaking, security, features, fixes}`.
`default_configuration`	object	no	Documented for parity; the deployed app's own `module_hooks.py` remains the actual source of truth.
`permissions`	object[]	no	Each `{codename, name, description}`, upserted into the RBAC catalog — registers the codename only, doesn't grant it to any role.
`is_active`	boolean	no	Forced `false` if no deployed code exists for `slug`.

Field	Type	Required	Notes
`permissions_schema`	object	**yes**	Maps this app's settings to the platform's scoped-override hierarchy (department/position/group/user). An upload with none at all fails validation.

A full example, `crm_lite`:

```
{
  "slug": "crm_lite",
  "name": "CRM Lite",
  "version": "1.0.0",
  "description": "Lightweight lead-tracking module.",
  "category": "Sales",
  "permissions": [
    { "codename": "crm_lite.lead.view", "name": "View leads" },
    { "codename": "crm_lite.lead.create", "name": "Create leads" }
  ],
  "permissions_schema": {
    "version": 1,
    "nodes": [
      { "key": "allow_lead_export", "default": false, "scopes": ["department",
"position", "group", "user"] },
      { "key": "allow_bulk_delete", "default": false }
    ]
  }
}
```

No `entry_point`, no file paths, no code of any kind — intentionally absent. This path is for the Sunesix platform team publishing modules whose real logic (models, services, views, permission enforcement) ships through an ordinary deploy under `backend/modules/<slug>/`. **If you're an external developer, this is not your path — skip to 2.2.**

2.2 THIRD-PARTY `PARTNERAPP` SUBMISSION

A third-party app is a row in `partner_apps.PartnerApp`, owned by a `DeveloperAccount` — a vendor identity, deliberately not a `Company`. Building an app for other companies to install doesn't make the vendor a Sunesix customer. There's no zip to upload; you build

a draft directly through the developer API/console.

Core fields:

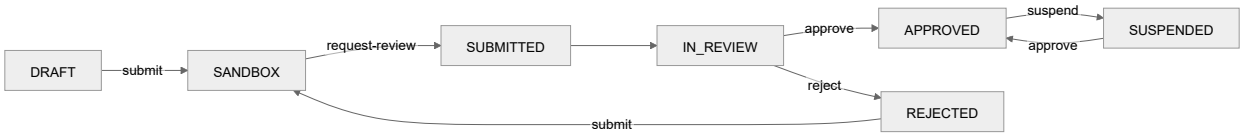
Field	Type	Notes
`call_direction`	enum	`INBOUND`, `OUTBOUND`, or `BOTH`.
`capabilities_requested`	string[]	Inbound codenames like `conversations.read` — see §3's capability table.
`outbound_endpoint_url`	string	Required if `call_direction` includes `OUTBOUND`.
`outbound_secret`	string	HMAC key Sunesix signs outbound requests with, so your endpoint can verify a call really came from Sunesix.
`failure_mode`	enum	`FAIL_OPEN` or `FAIL_CLOSED` — what happens if your OUTBOUND endpoint is unreachable.
`version`, `changelog`, `min_platform_version`	—	Same shape `modules.Module` already uses.
`allowed_models`	string[]	Set by a Sunesix reviewer at approval, not by you — clamps which models `ai.invoke` may call.
`distribution`	enum	`PRIVATE` (default) or `MARKETPLACE`.

Where this lands in the catalog. Approval upserts a `Module` catalog row directly (`is_active=true`, `is_system_module=false`) — the exact same table row shape §2.1 describes, just created by a different path. `ModuleManager` never checks for deployed code; only the manifest-upload pipeline does. So a `PartnerApp`-backed catalog entry works completely unmodified with the existing install/enable/disable/dependency/RBAC/nav machinery — structurally, it's just a normal `Module` row whose "code" happens to run entirely on your own infrastructure.

---

3. Building a third-party app: step by step

Every app moves through the same five states, in this order. Nothing in this guide describes a shortcut around it.



3.1 REGISTER A DEVELOPER ACCOUNT

A developer account is a vendor identity, separate from a company account. Two signup shapes share one endpoint: if the caller is an authenticated Company Admin, their company is attached automatically and the account becomes **BUSINESS**; otherwise it's **INDEPENDENT**. The only difference is which fields get pre-filled — the review pipeline is identical either way.

```
POST /partner-apps/developer/register/
{
  "name": "Acme Guardrails",
  "contact_email": "dev@acme.io",
  "admin_username": "acme-dev",
  "admin_email": "dev@acme.io",
  "legal_name": "Acme Guardrails, Inc.",
  "website": "https://acme.io",
  "country": "US",
  "tax_id": "12-3456789"
}
```

The response includes a one-time `temporary_password` for your developer login — it isn't shown again. Sign in at the developer portal (`POST /auth/developer/token/`), not the company or admin one.

Your account starts `PENDING_VERIFICATION`. You can create and edit app drafts immediately; you just can't put anything in front of a customer until a Sunesix reviewer approves your account — checked in the service layer, not the view, so there's no path around it.

3.2 DECLARE WHAT YOUR APP DOES

Create a draft with the capabilities and call direction it needs. Request only what you actually use — every capability maps to the same data a Company Admin already sees in their own UI, nothing more.

```
POST /partner-apps/developer/apps/
{
  "name": "Acme Prompt Guard",
  "call_direction": "OUTBOUND",
  "capabilities_requested": [],
  "outbound_endpoint_url": "https://guard.acme.io/hooks/verdict",
  "failure_mode": "FAIL_CLOSED",
  "version": "0.1.0"
}
```

Inbound capabilities:

Capability	Read scope
`conversations.read`	AI conversation history for the installing company.
`employees.read`	Employee profiles.
`usage.read`	Token usage series.
`ai.invoke`	Call the gateway to run inference — see §4.2.

A draft is yours alone; nothing here is visible to any company yet.

3.3 TEST IN THE SANDBOX

```
POST /partner-apps/developer/apps/{id}/submit/
```

Moves your app from `DRAFT` (or `REJECTED`, after a fix) to `SANDBOX`, and provisions it against a reserved, synthetic tenant — fake employees, conversations, and usage, never a real company's data. This is a **data-reach** sandbox, not a code sandbox: nothing of yours ever runs on Sunesix infrastructure, so there's nothing to contain at the process level. The guarantee is entirely about which tenant your key can reach.

```
POST /partner-apps/developer/apps/{id}/sandbox-key/
→ { "success": true, "data": { "api_key": "pak.42.7f3a9c1e8b6d4f2a..." } }
```

The raw key is shown exactly once. Losing it isn't a recovery problem — mint again and the old one is immediately revoked. Use this key against the capability endpoints (§4.1) or gateway inference (§4.2), and to exercise your own outbound endpoint if you're building a guardrail. Any app that isn't `APPROVED` can only ever authenticate with a sandbox-scoped key — that check runs at authentication, not per-capability, so every inbound surface inherits it automatically, including ones added later.

3.4 CERTIFICATION & REVIEW

```
POST /partner-apps/developer/apps/{id}/request-review/
```

Moves `SANDBOX` → `SUBMITTED`, then a reviewer marks it `IN_REVIEW`. This is refused if your developer account isn't verified yet (§3.1). A reviewer runs five probes against your sandbox tenant — approval is refused until every one of them passes:

Probe	Fails when
`capabilities_declared`	You requested a capability the platform doesn't implement.
`capability:<name>`	A requested read handler errors against sandbox data.
`inference_allowlist`	`ai.invoke` requested with no `allowed_models`, or a model not in the registry.
`outbound_handshake`	Your endpoint is unreachable, or its response has no `allow` key.
`sandbox_confinement`	An active key for your app exists outside the sandbox tenant.

```
GET /partner-apps/developer/apps/{id}/certification/
→ {
    "passed_at": null,
    "results": [
      { "probe": "capabilities_declared", "passed": true },
      { "probe": "outbound_handshake", "passed": false, "detail": "connection
timed out" }
    ]
  }
```

Probes report structured pass/fail rather than raising — a reviewer needs to see *which* check failed, and your endpoint being briefly down during certification is an expected outcome, not a fatal one. If rejected, fix the issue and submit again: a rejected app goes straight back through `SANDBOX`, no need to recreate the draft.

3.5 GO LIVE

Approval creates your app's first real catalog listing — nothing before this point was visible to any company. Choose distribution: **Private** (default — the common case for a business connecting its own agent) or **Marketplace** (public listing). A private app installs only for your own company (if you registered as a business) plus any company you've explicitly invited.

Your sandbox key still only ever reaches the sandbox tenant — that never changes. Once a real company installs your app, they issue you a separate, production-scoped key for that specific install; you'll hold one key per installing company, not one key for your whole app.

A Sunesix reviewer can suspend an approved app at any time — every install stops working immediately, platform-wide. This is distinct from a single company disabling its own install via the ordinary `ModuleManager.disable()`.

---

4. API reference

4.1 CAPABILITY API

Every request: `Authorization: Bearer pak.<id>.<secret>`. The installing company is always resolved for you from the key — there's no company parameter to pass or spoof.

```
GET /partner-apps/capabilities/conversations/
→ { "success": true, "data": [
      { "id": 91, "employee_id": 12, "started_at": "2026-08-01T14:02:00Z",
"message_count": 6 }
    ] }

GET /partner-apps/capabilities/employees/
→ { "success": true, "data": [
      { "id": 12, "full_name": "Jordan Lee", "department": "Support",
"position": "Agent" }
    ] }

GET /partner-apps/capabilities/usage/
→ { "success": true, "data": [
      { "date": "2026-08-01", "tokens": 48210, "requests": 312 }
    ] }
```

Inbound partner traffic has its own rate budget — **300 requests/hour** per key — independent of the installing company's own human-traffic limit, so a busy integration never eats into their normal usage allowance.

4.2 GATEWAY INFERENCE

```
POST /partner-apps/ai/chat/
{ "model_id": "claude-sonnet-5", "prompt": "Summarize this week's flagged
conversations." }

→ { "success": true, "data": {
      "text": "...", "model": "claude-sonnet-5", "provider": "anthropic",
      "usage": { "input_tokens": 812, "output_tokens": 140 }
    } }
```

Requires the `ai.invoke` capability, and `model_id` must be in your app's `allowed_models` list — a Sunesix reviewer sets that at approval, not you. This is the *same* funnel every human chat turn goes through, one hop fewer, not a second, ungoverned path: spend limits, prompt-injection defense, output guardrails, and usage logging all apply exactly as they do to a person typing in the product. This is also what lets your agent use Sunesix models without holding a provider credential of your own — apps receive data, never keys.

Whether the installing company's balance or your own developer balance is charged is an inference-billing setting a reviewer sets per app, alongside a monthly cap. A developer-pays app burns real tokens whatever its retail price is, including a free one — check your billing mode before assuming a free listing costs you nothing.

4.3 OUTBOUND GUARDRAIL HOOK

If your app is `OUTBOUND`-capable, Sunesix calls your endpoint the mirror-image way. Every request carries an `X-Insights-Signature` header — HMAC-SHA256 over the body, using your `outbound_secret`. Verify it before trusting a call claims to come from Sunesix.

```
Your endpoint receives:
{ "company_id": 4, "prompt": "...", "phase": "input" }

Your endpoint returns:
{ "allow": true }
```

On an unreachable endpoint, `failure_mode` decides the outcome: `FAIL_OPEN` (allow, log nothing beyond the swallowed exception) or `FAIL_CLOSED` (block the request) — a policy you set at submission time, reviewed by Sunesix, never silently hardcoded to one behavior.

---

5. Things that are easy to get wrong

- **There is no code upload for third-party apps, ever.** If you're picturing a zip with your service's source in it, that's the §2.1 first-party path — it doesn't apply to you, and it never will for a third-party integration by design.

- **Your sandbox key never becomes your production key.** Each installing company mints you a separate key when they install. Don't hardcode the sandbox key anywhere that will still be running after launch.
- `capabilities_requested` **is a ceiling, not a suggestion.** Requesting more than you use doesn't get you more access later — it gets your submission flagged during review for scope you can't justify.
- **A rejected app doesn't restart from** `DRAFT`**.** It goes back to `SANDBOX`, so your existing sandbox key and test setup still work while you fix whatever failed.
- `FAIL_OPEN` **vs** `FAIL_CLOSED` **is a real security decision, not a formality.** If your guardrail goes down, `FAIL_OPEN` means every request sails through unchecked. Pick deliberately.