SYRO ONE — Content Master API Reference

> **Audience:** the frontend integrator wiring `knorg-syro` (SYRO ONE) to these backend APIs. **Owner:** `k-organisation-admin-backend`. Built phase-by-phase from `content-master-spec.md`. **This doc is updated in the same change set as each phase's code** — if an endpoint is listed below, it is built and live in the codebase (subject to the gateway routing note).
>
> Sync target: deltas from the original spec are noted per-section and should be mirrored back to `../../knorg-syro/docs/syro/CONTENT-MASTER-SPEC.md`.

Coverage

#	Master	Status	Base path
5–7	Page CMS (About / History / Faith / any explainer page)	✅ **built** (Phase 2)	`/site/pages`
9	Saints & Blesseds	✅ **built** (Phase 1)	`/site/saints`
16	Announcements (global / eparchy + parish aggregate)	✅ **built** (Phase 3)	`/site/announcements`
15	Events + RSVP (global / eparchy / non-ERP parish)	✅ **built** (Phase 4)	`/site/events`
8	Liturgical Calendar (seasons / feasts / fasts)	✅ **built** (Phase 5)	`/site/calendar`
10	Pastoral Letters (global / eparchy authored)	✅ **built** (Phase 6)	`/site/pastoral-letters`
11	Circulars (global / eparchy authored)	✅ **built** (Phase 7)	`/site/circulars`
12	News + Press (news: global/eparchy/non-ERP parish; press: global/eparchy)	✅ **built** (Phase 8)	`/site/news`, `/site/press`
13	Photo Gallery (albums + images; global/eparchy/non-ERP parish)	✅ **built** (Phase 9)	`/site/gallery`
14	Services / Mass Timings (read-only aggregation over parish spine)	✅ **built** (Phase 10)	`/site/services`
1–4	Site singletons — Navigation / Settings / Footer / Home	✅ **built** (Phase 11)	`/site/{navigation,settings,footer,home}`

#	Master	Status	Base path
18	Parish Website Content (per-parish micro-site)	✅ **built** (Phase 12)	`/parish/{parishId}/website-content`
17	Dashboards (global / regional / parish / me)	✅ **built** (Phase 13)	`/site/dashboard`

All 18 modules built. 🎉

Conventions (apply to every master)

- **Response envelope** — every endpoint returns the repo-standard ad-hoc map:

  ```
  { "error": false, "statusCode": 200, "data": <payload>, "message": "..." }
  ```

  `statusCode` in the body mirrors the HTTP status. On error, `error=true` and `message` carries the reason; `data` is absent.

- **Provider-scope, NOT tenant-scope.** SyroOne content masters have **no `clientid`**. Do **not** send `clientid` — it is ignored. (This is the deliberate split from the legacy tenant `manage-*` DynamoDB APIs, which are a different product surface for ERP tenants.)

- **Vertical dimension** — every record also carries a **`verticalCode`** (e.g. `SYRO-MALABAR`). This is the platform vertical the content belongs to; **GLOBAL scope = the vertical itself**, so a future second vertical's website has its own global content.

  - On **create/update**: `verticalCode` in the body is used for **global** records (defaults to `SYRO-MALABAR` when blank). For **eparchy/parish** records it is **derived from the referenced node/parish** (authoritative) — the body value is ignored.
  - On **reads**: every list / list-published / published / aggregate / dashboard / mass-timing endpoint accepts a `vertical` query param (`required=false`, default `SYRO-MALABAR`) and filters results to that vertical. By-id reads (`/published/{id}`, `/{id}`) take **no** `vertical` param — the id is globally unique. With a single vertical today the filter is a functional no-op, but it is wired end-to-end (controller → service → JPA Specification → `vertical_code` column) so a second vertical works without further backend changes.

- **Audit columns** — every content-master table carries four audit fields, surfaced on every response object: `createdAt` / `modifiedAt` (timestamps, set automatically by Hibernate on insert/update) and `createdBy` / `updatedBy` (the acting user). `createdBy`/ `updatedBy` are populated by JPA auditing from the request principal — the gateway-injected `X-User-Id` (`AuthorizationContext.getPrincipalId()`); they are **null** when no principal is present (e.g. an unauthenticated read or a background scheduler). `createdBy` is write-once (never changed after insert). Clients do **not** send these — any value in the body is ignored.

- **The scope model** — every scoped record carries three fields:

Field	Wire type	Values	Meaning
`scope`	string	`global` | `eparchy` | `parish`	Level the record lives at. `regional` is accepted as an alias for `eparchy`.
`ownerType`	string	`platform` | `eparchy` | `parish`	What owns it. Inferred from `scope` if omitted.
`ownerId`	number (Long) | null	—	**Polymorphic, resolved by `ownerType`** (see below).

`ownerId` resolution (important — it is NOT a `clientid`):

scope	ownerType	ownerId
`global`	`platform`	must be **null**
`eparchy`	`eparchy`	a **`template_hierarchy_nodes.id`** (Long)
`parish`	`parish`	a **`parish.id`** (Long, from `/parish/*`)

The server validates the triple on create/update and returns **400** if `ownerType` conflicts with `scope`, if `ownerId` is missing for eparchy/parish scope, if `ownerId` is present for global scope, or if `ownerId` does not reference an existing eparchy/parish.

- **IDs** — id type is **per master**: typed masters near the parish spine use **numeric (Long) ids** (e.g. Saints); the generic **Page CMS uses string UUID ids** (matching the existing `manage_pages` CMS, and the spec's "CMS content uses string ids"). Each master's section states its id type. Treat them as opaque JSON values.
- **Icons** — any icon field is a string `iconKey` (e.g. `"heart"`); never icon markup.
- **Auth / gateway** — these are provider/public endpoints; the backend does not enforce auth (gateway-trust posture). **Routing is wired** in `k-org-api-gateway` (migration `V23__Add_syro_site_content_route.sql`): the gateway forwards **`/site/**`** → **admin-backend with no prefix strip** (same bare-path convention knorg-syro already uses for `/parish/*` and `/institutions/*`). Call these endpoints at the gateway base URL (`KORG_ADMIN_API_BASE_URL`, prod `https://korgservice.com`) using the path exactly as written below — **no `/korg` or `/local` prefix**.
  - **Public (no JWT):** each master's **`list-published`** read is added to the gateway public-route allowlist per phase (Saints: `/site/saints/list-published`). knorg-syro's `x-api-key` is accepted because the path bypasses auth entirely.
  - **Authenticated (JWT required):** every other `/site/**` path — the admin/full list, get-by-id, and all writes (`POST`/`PUT`/`PATCH`/`DELETE`) — still passes through the gateway's `RequireAuthorizationHeader` filter and needs a valid bearer token. (Intended for a future SyroOne admin tool; the public site only needs `list-published`.)

Module 9 — Saints & Blesseds *(Phase 1 — built)*

Powers the public `/saints` page on knorg-syro (the `saints[]` list). The page's framing content (`hero`, canonization `process[]`, `prayer`, `cta`) is **not** part of this master — that belongs to the generic Page CMS (a later phase).

> **Delta from spec:** the spec suggested reusing `GET /manage-saints/list-published`. That path is the **tenant DynamoDB** Saints API (ERP clients) and is left untouched. SyroOne Saints live at the new provider-scope path `/site/saints`.

Resource shape (`data`)

```
{
  "id": 12,                        // Long, server-assigned
  "name": "St. Thomas the Apostle",
  "type": "Saint",                 // "Saint" | "Blessed" | "Venerable"  (default
"Saint")
  "feast": "July 3",               // free text, nullable
  "years": "1st century",          // free text life span, nullable
  "image": "https://.../thomas.jpg", // URL, nullable
  "bio": "Apostle to India ...",   // long text, nullable
  "scope": "global",               // see scope model above (default "global")
  "ownerType": "platform",
  "ownerId": null,
  "isPublished": false,            // default false
  "sortOrder": 0,                  // ascending; default 0
  "createdAt": "2026-06-04T10:15:00.000+00:00",
  "modifiedAt": "2026-06-04T10:15:00.000+00:00"
}
```

Saints are almost always `scope=global`. Eparchy/parish scoping is supported by the same fields if a locally-venerated saint is ever needed.

Endpoints

Method	Path	Purpose
`GET`	`/site/saints`	Full list (published **and** unpublished) — admin view.
`GET`	`/site/saints/list-published`	Published rows only — public consumption.
`GET`	`/site/saints/{id}`	Single record.
`POST`	`/site/saints`	Create.
`PUT`	`/site/saints/{id}`	Update (full replace of mutable fields).
`PATCH`	`/site/saints/{id}/publish`	Set `isPublished=true`.
`PATCH`	`/site/saints/{id}/unpublish`	Set `isPublished=false`.
`DELETE`	`/site/saints/{id}`	Hard delete.

Auth: only `GET /site/saints/list-published` is public (gateway allowlist). All other rows above require a valid JWT through the gateway.

Query params (both `GET` list endpoints — all optional):

Param	Type	Notes
`scope`	string	`global` / `eparchy` / `parish` (or `regional`).
`ownerType`	string	`platform` / `eparchy` / `parish`.
`ownerId`	number	Long.
`type`	string	`Saint` / `Blessed` / `Venerable`.
`publishedOnly`	boolean	`GET /site/saints` only (default `false`). The `/list-published` endpoint forces `true`.

Results are ordered by `sortOrder` ascending, then `name` ascending.

Examples

Create a global saint (draft):

```
POST /site/saints
Content-Type: application/json

{ "name": "St. Alphonsa", "type": "Saint", "feast": "July 28", "years": "1910–
1946",
  "bio": "First woman of Indian origin to be canonized.", "sortOrder": 1 }
```

Publish it:

```
PATCH /site/saints/12/publish
```

Public list (published only, blesseds):

```
GET /site/saints/list-published?type=Blessed
```

Validation / error notes

- `name` is required → 400 if missing/blank.
- Unknown `scope` / `ownerType` / `type` value → 400 with an explanatory `message`.
- Scope/owner inconsistency or non-existent `ownerId` → 400 (see scope model above).
- `GET`/`PATCH`/`DELETE` on a missing id → 404.

Modules 5–7 — Page CMS (About / History / Faith / any explainer page) *(Phase 2 — built)*

ONE generic page master backs every marketing/explainer page instead of a bespoke table per page. A page = a shared **hero** + an ordered list of typed **sections** + a **cta** block. The frontend maps each section's `type` string to a component (just like `iconKey`) and reads its data from `props` — so adding a new page or block type needs **no backend change**.

> **Delta from spec:** the spec listed per-page endpoints (`GET /site/pages/about`, `/history`, `/faith`). They are unified into one master keyed by `slug`; the public read is **`GET /site/pages/published/{slug}`** (so `about` → `/site/pages/published/about`). About/History/Faith are just three rows with those slugs.

Resource shape (`data`)

```
{
  "id": "8f3a…-uuid",              // string UUID, server-assigned
  "slug": "about",                 // lowercase kebab-case; unique within a
scope/owner
  "title": "About SYRO ONE",
  "hero": {                        // nullable
    "title": "About Us",
    "breadcrumb": ["Home", "About"],
    "backgroundImage": "https://.../about-hero.jpg"
  },
  "sections": [                    // ordered; each block is type + open-ended
props
    { "id": "s1", "type": "stats",   "order": 1,
      "props": { "items": [ { "value": "120+", "label": "Parishes" } ] } },
    { "id": "s2", "type": "mission", "order": 2, "props": { "text": "…" } },
    { "id": "s3", "type": "values",  "order": 3,
      "props": { "items": [ { "iconKey": "heart", "title": "Faith", "body": "…" }
] } },
    { "id": "s4", "type": "timeline","order": 4,
      "props": { "items": [ { "year": "1887", "text": "…" } ] } }
  ],
  "cta": {                         // nullable
    "eyebrow": "Join us", "title": "Find a parish", "subtitle": "…",
    "buttons": [ { "label": "Directory", "href": "/directory", "style": "primary"
} ]
  },
  "scope": "global", "ownerType": "platform", "ownerId": null,
  "isPublished": false, "sortOrder": 0,
  "createdAt": "2026-06-04T10:15:00.000+00:00",
  "modifiedAt": "2026-06-04T10:15:00.000+00:00"
}
```

- `hero`, `sections[].props`, and `cta` are stored as JSONB and round-trip verbatim — the block/prop shapes are the contract owned jointly by the SyroOne frontend and the admin tool. The server validates only the envelope (`slug`, `title`, scope), never the `props` content.

- `type` is an **open string** (e.g. `stats`, `mission`, `vision`, `values`, `timeline`, `pillars`, `beliefs`, `sacraments`, `traditions`, `feature`). No server-side catalog.

Endpoints

Public (no JWT — gateway-allowlisted, read-only):

Method	Path	Purpose
`GET`	`/site/pages/published`	All published pages (any scope), full docs, ordered.
`GET`	`/site/pages/published/{slug}`	One published **global** page by slug (about/history/faith). 404 if absent/unpublished.

Management (JWT required):

Method	Path	Purpose
`GET`	`/site/pages`	Full list incl. drafts. Optional `scope` / `ownerType` / `ownerId` / `publishedOnly`.
`POST`	`/site/pages`	Create.
`PUT`	`/site/pages/{id}`	Update (full replace of mutable fields; `{id}` is the UUID).
`PATCH`	`/site/pages/{id}/publish`	Set `isPublished=true`.
`PATCH`	`/site/pages/{id}/unpublish`	Set `isPublished=false`.
`DELETE`	`/site/pages/{id}`	Hard delete.

Example

Create the About page (draft, global):

```
POST /site/pages
Content-Type: application/json

{ "slug": "about", "title": "About SYRO ONE",
  "hero": { "title": "About Us", "breadcrumb": ["Home","About"] },
  "sections": [ { "id": "s1", "type": "mission", "order": 1, "props": { "text":
"…" } } ],
  "cta": { "title": "Find a parish", "buttons": [ { "label": "Directory", "href":
"/directory" } ] } }
```

Then `PATCH /site/pages/{id}/publish`, and the public site reads `GET /site/pages/published/about`.

Validation / error notes

- `slug` and `title` are required → 400 if missing/blank. `slug` is lowercased server-side.
- Duplicate `slug` within the same scope/owner → **409**.
- Scope/owner rules are identical to the shared model above (400 on violation).

- Unknown `scope` / `ownerType` query value → 400. Missing published page → 404.

Module 16 — Announcements *(Phase 3 — built)*

Scoped announcements with the full global / eparchy / parish resolution. **Important split:** this master (`syro_announcement`) holds **GLOBAL and EPARCHY** announcements only. **PARISH announcements keep living in the existing `parish_announcement` table** and are served by the parish spine at `GET /parish/{parishId}/announcements` (which knorg-syro already uses) — there is **no second parish store**. The **aggregate** endpoint merges both sources into one feed; because the two stores cover disjoint scopes, **de-dup is inherent — a row never appears twice** (the spec's "define as a query/policy rule, not duplicate rows").

> **Delta from spec:** the spec sketched extending `GET /manage-announcements/list-published` (the tenant DynamoDB API). Instead, global/eparchy live at the new provider-scope `/site/announcements`, and the cross-scope feed is `GET /site/announcements/aggregate`.

- **id type:** numeric **Long** (typed master). In the aggregate feed, ids are **not unique across sources** — always pair an id with its `source` (`"site"` vs `"parish"`).
- Allowed `scope` here is **`global` / `eparchy`** only; posting `parish` → **400** pointing you to `/parish/{parishId}/announcements`.

Resource shape (`data` — from `/site/announcements*`)

```
{
  "id": 7,
  "title": "Lenten retreat schedule",
  "content": "…",                 // long text, nullable
  "type": "liturgical",           // open category string, nullable
  "publishedDate": "2026-06-04",  // ISO date; defaults to today on create
  "expiryDate": "2026-07-01",     // ISO date, nullable (null = never expires)
  "isPinned": false,
  "scope": "eparchy",             // "global" | "eparchy"
  "ownerType": "eparchy",         // "platform" | "eparchy"
  "ownerId": 42,                  // eparchy = template_hierarchy_nodes.id; null
when global
  "isPublished": false,
  "sortOrder": 0,
  "createdAt": "…", "modifiedAt": "…"
}
```

Endpoints

Public (no JWT — gateway-allowlisted, read-only):

Method	Path	Purpose
`GET`	`/site/announcements/list-published`	Published + non-expired global/eparchy rows. Filters: `scope`, `ownerType`, `ownerId`, `type`. Pass

Method	Path	Purpose
		`scope=global` for the SyroOne home page; `scope=eparchy&ownerId=42` for an eparchy page.
`GET`	`/site/announcements/aggregate`	Merged feed: **global (always)** + the given eparchy + the given parish. Params: `eparchyId`, `parishId` (both optional). Returns `AnnouncementView[]` (see below).

Management (JWT required):

Method	Path	Purpose
`GET`	`/site/announcements`	Full list incl. drafts/expired. Filters: `scope` / `ownerType` / `ownerId` / `type` / `publishedOnly`.
`POST`	`/site/announcements`	Create (scope `global`/`eparchy` only).
`PUT`	`/site/announcements/{id}`	Update.
`PATCH`	`/site/announcements/{id}/publish` | `/unpublish`	Toggle publish.
`DELETE`	`/site/announcements/{id}`	Hard delete.

All listings are ordered **pinned first, then `publishedDate` desc, then `id` desc**.

Aggregate feed item (`AnnouncementView`)

```
{
  "source": "parish",            // "site" (syro_announcement) | "parish"
(parish_announcement)
  "id": 3,                       // unique only WITHIN its source
  "title": "Parish feast day",
  "content": "…",
  "type": null,                  // parish rows have no type
  "scope": "parish",             // "global" | "eparchy" | "parish"
  "ownerType": "parish",
  "ownerId": 11,                 // parish.id for parish rows; eparchy node id for
eparchy rows; null for global
  "publishedDate": "2026-06-01",
  "expiryDate": null,
  "isPinned": true
}
```

Example

```
POST /site/announcements
Content-Type: application/json
```

```
{ "title": "Lenten retreat", "content": "…", "type": "liturgical",
  "scope": "eparchy", "ownerId": 42, "expiryDate": "2026-07-01", "isPinned": true
}
```

Eparchy + parish combined feed for a parish page:

```
GET /site/announcements/aggregate?eparchyId=42&parishId=11
```

Validation / error notes

- `title` is required → 400. `scope=parish` (or `ownerType=parish`) → 400 (use the parish spine).
- Eparchy `ownerId` must reference an existing `template_hierarchy_nodes.id` → 400 otherwise.
- Unknown `scope` / `ownerType` / enum value → 400. Missing id on mutate → 404.

Module 15 — Events + RSVP *(Phase 4 — built)*

Scoped events with member RSVP. Holds **global / eparchy / non-ERP-parish** events.

> **Two event sources — important.** There is a separate **tenant ERP Events Module** (`/events`, keyed by `clientid`). **ERP-client parishes manage their events there**, and knorg-syro reads an ERP parish's events from that module directly. This master is for the SyroOne (global), eparchy, and **non-ERP (directory-only) parish** levels. Posting a `scope=parish` event whose `ownerId` is an **ERP-client parish** (`parish.is_erp_client = true`) → **400**, pointing you to the ERP module. So per parish: ERP parish ⇒ `/events` (tenant module); non-ERP parish ⇒ `/site/events` here.

> **Delta from spec:** the spec used the bare `/events` path; that belongs to the ERP module, so this master lives at **`/site/events`** and the public single read is **`/site/events/published/{id}`**.

- **id type:** numeric **Long** (both events and RSVPs).
- Timestamps (`startAt` / `endAt`) are **ISO-8601 instants in UTC** (e.g. `2026-06-04T18:30:00Z`).

Resource shape (`data`)

```
{
  "id": 5,
  "title": "Parish feast celebration",
  "description": "…",
  "eventType": "feast",          // open category string (frontend maps to
EVENT_TYPES)
  "startAt": "2026-07-03T09:00:00Z",
  "endAt":   "2026-07-03T12:00:00Z",  // must not precede startAt
  "location": "St. Thomas Church, Kochi",
  "imageUrl": "https://.../feast.jpg",
  "capacity": 300,               // nullable = uncapped
  "scope": "parish",             // "global" | "eparchy" | "parish"
  "ownerType": "parish",
```

```
  "ownerId": 11,                  // parish.id (non-ERP) | eparchy node id | null
(global)
  "isPublished": false,
  "sortOrder": 0,
  "createdAt": "…", "modifiedAt": "…"
}
```

Endpoints

Public (no JWT — gateway-allowlisted):

Method	Path	Purpose
GET	/site/events/list-published	Published events. Filters: scope, ownerType, ownerId, type, from, to (ISO instants). Parish view: scope=parish&ownerId=11; SyroOne: scope=global.
GET	/site/events/published/{id}	One published event. 404 if absent/unpublished.

Management / authenticated (JWT required):

Method	Path	Purpose
GET	/site/events	Full list incl. drafts. Same filters as above + publishedOnly.
POST	/site/events	Create.
PUT	/site/events/{id}	Update.
PATCH	/site/events/{id}/publish | /unpublish	Toggle publish.
DELETE	/site/events/{id}	Hard delete (cascades to RSVPs).
POST	/site/events/{id}/rsvp	Upsert the member's RSVP (one per event+user). **Identified users only** — see RSVP note.
GET	/site/events/{id}/rsvp-summary	RSVP counts for the event.
GET	/site/events/{id}/rsvps	All RSVP rows for the event.

Listings are ordered by startAt ascending (soonest first; undated events last).

RSVP — identified users only (flow pending)

> **RSVP is NOT public.** A member may RSVP only once the system has identified them, so the RSVP endpoints are **not** on the gateway public allowlist — they require a valid JWT. The **member-login method and the end-to-end RSVP flow are still to be defined**; the endpoints below are built and JWT-gated, but the wiring of userId to the authenticated identity (vs. the request body) will be finalized when that login method lands. Treat this surface as provisional.

POST /site/events/{id}/rsvp body:

```
{ "userId": "member-abc", "status": "going", "guests": 2 }
```

- `status`: `going` | `maybe` | `not_going` (defaults to `going`). `guests` defaults to `0`.
- Idempotent per `(eventId, userId)` — re-posting updates the existing RSVP.
- *(Pending:* `userId` will come from the identified user, not the body, once member login is defined.*)*

`GET /site/events/{id}/rsvp-summary` →

```
{ "eventId": 5, "going": 40, "maybe": 7, "notGoing": 3, "totalResponses": 50,
"goingGuests": 18 }
```

Validation / error notes

- `title` required → 400. `endAt` before `startAt` → 400. Negative `capacity`/`guests` → 400.
- `scope=parish` on an **ERP-client parish** → 400 (use the ERP `/events` module).
- Eparchy/parish `ownerId` must reference an existing node/parish → 400 otherwise.
- RSVP without `userId` → 400. Missing event/id → 404.

Module 8 — Liturgical Calendar *(Phase 5 — built)*

Three typed sub-masters behind one `/site/calendar` feature: **seasons**, **feasts**, **fasts**. Predominantly global (church-wide); the shared scope model is present for the occasional eparchy-local item. The calendar page's hero/cta framing is a **Page CMS** row (Phase 2, slug `calendar`) — not part of this master.

> **Delta from spec:** the spec sketched `/manage-liturgical-seasons|feasts|fasts`. Unified under `/site/calendar/{seasons,feasts,fasts}`, plus one combined public read.

- **id type:** numeric **Long** for all three.

Resource shapes (`data`)

```
// season
{ "id": 1, "name": "Great Lent", "period": "Feb 16 – Apr 4", "iconKey": "cross",
  "styleKey": "purple", "body": "…", "scope": "global", "ownerType": "platform",
  "ownerId": null, "isPublished": true, "sortOrder": 3, "createdAt": "…",
"modifiedAt": "…" }
// feast
{ "id": 9, "name": "Dukrana of St. Thomas", "month": 7, "day": 3, "description":
"…",
  "scope": "global", "ownerType": "platform", "ownerId": null, "isPublished":
true,
  "sortOrder": 0, "createdAt": "…", "modifiedAt": "…" }
// fast
{ "id": 4, "name": "Great Lent", "duration": "50 days", "note": "…", "scope":
"global",
```

```
  "ownerType": "platform", "ownerId": null, "isPublished": true, "sortOrder": 0,
... }
```

`month` (1–12) and `day` (1–31) are nullable (movable feasts). `iconKey` / `styleKey` are strings the frontend maps.

Endpoints

Public (no JWT — gateway-allowlisted):

Method	Path	Purpose
`GET`	`/site/calendar/published`	The whole published calendar in one call. Optional `scope` / `ownerType` / `ownerId`. Returns `{ "seasons": [...], "feasts": [...], "fasts": [...] }`.

Management (JWT required) — identical CRUD for each of `seasons`, `feasts`, `fasts`:

Method	Path	Purpose
`GET`	`/site/calendar/{type}`	List incl. drafts. Filters `scope` / `ownerType` / `ownerId` / `publishedOnly`.
`POST`	`/site/calendar/{type}`	Create.
`PUT`	`/site/calendar/{type}/{id}`	Update.
`PATCH`	`/site/calendar/{type}/{id}/publish` | `/unpublish`	Toggle publish.
`DELETE`	`/site/calendar/{type}/{id}`	Hard delete.

`{type}` ∈ `seasons` | `feasts` | `fasts`. Seasons & fasts sort by `sortOrder` then `name`; feasts sort by `month`, `day` (calendar order).

Validation / error notes

- `name` required → 400. Feast `month` must be 1–12, `day` 1–31 → 400 otherwise.
- Scope/owner rules per the shared model. Unknown enum value → 400. Missing id → 404.

Module 10 — Pastoral Letters *(Phase 6 — built)*

The real letters list behind the `/pastoral` page. **Global or eparchy authored only** (a pastoral letter is issued church-wide or by an eparchy — never at parish level; `scope=parish` → 400). The page framing (hero / intro / `themes[]` / `excerpts[]` / cta) is a **Page CMS** row (Phase 2, slug `pastoral`) — read it from `GET /site/pages/published/pastoral`.

> **Delta from spec:** the spec sketched `/manage-pastoral-letters/list-published`. The letters live at the provider-scope `/site/pastoral-letters`.

- **id type:** numeric **Long**.

Resource shape (`data`)

```
{
  "id": 3,
  "title": "On the Year of Faith",
  "author": "Mar George Alencherry",
  "role": "Major Archbishop",
  "publishedDate": "2026-01-06",  // ISO date; defaults to today on create
  "summary": "…",                 // short text, nullable
  "bodyHtml": "<p>…</p>",         // rich text, nullable
  "attachmentUrl": "https://.../letter.pdf",  // nullable
  "scope": "global",              // "global" | "eparchy"
  "ownerType": "platform",        // "platform" | "eparchy"
  "ownerId": null,                // eparchy = template_hierarchy_nodes.id; null
when global
  "isPublished": false,
  "sortOrder": 0,
  "createdAt": "…", "modifiedAt": "…"
}
```

Endpoints

Public (no JWT — gateway-allowlisted):

Method	Path	Purpose
`GET`	`/site/pastoral-letters/list-published`	Published letters. Filters: `scope`, `ownerType`, `ownerId`. `scope=global` church-wide; `scope=eparchy&ownerId=42` for an eparchy.
`GET`	`/site/pastoral-letters/published/{id}`	One published letter (with `bodyHtml`). 404 if absent/unpublished.

Management (JWT required):

Method	Path	Purpose
`GET`	`/site/pastoral-letters`	Full list incl. drafts. Filters + `publishedOnly`.
`POST`	`/site/pastoral-letters`	Create.
`PUT`	`/site/pastoral-letters/{id}`	Update.
`PATCH`	`/site/pastoral-letters/{id}/publish` | `/unpublish`	Toggle publish.
`DELETE`	`/site/pastoral-letters/{id}`	Hard delete.

Listings are ordered by `publishedDate` desc, then `id` desc.

Validation / error notes

- `title` required → 400. `scope=parish` (or `ownerType=parish`) → 400 (global/eparchy only).
- Eparchy `ownerId` must reference an existing `template_hierarchy_nodes.id` → 400 otherwise.
- Unknown enum value → 400. Missing id → 404.

Module 12 — News + Press *(Phase 8 — built)*

Two masters: **News** (`/site/news`) and **Press releases** (`/site/press`).

News

Global / eparchy / **non-ERP parish** news. Like Events, **ERP-client parishes manage their news in the tenant module** (`/manage-news`, keyed by `clientid`) — knorg-syro reads an ERP parish's news from there. Posting a `scope=parish` news item whose `ownerId` is an ERP-client parish → **400**. Field names mirror the frontend `WebsiteNews` shape so the type largely carries over — **but `id` is a numeric Long here** (not a string) and there is **no `clientid`** (scope/owner replace it).

> **Delta from spec:** the spec suggested `GET /manage-news/published?scope=global` (the tenant DynamoDB API). SyroOne news lives at the provider-scope `/site/news`.

```
// news item (data)
{ "id": 14, "headline": "New eparchy announced", "newsdate": "2026-05-30",
  "newsdescription": "…", "imageURL": "https://.../n.jpg", "categoryname":
"general",
  "scope": "global", "ownerType": "platform", "ownerId": null,
  "isPublished": false, "sortOrder": 0, "createdAt": "…", "modifiedAt": "…" }
```

Method	Path	Auth	Purpose
`GET`	`/site/news/list-published`	public	Published news. Filters: `scope`, `ownerType`, `ownerId`, `category`. Parish view: `scope=parish&ownerId=11`.
`GET`	`/site/news/published/{id}`	public	One published item.
`GET`	`/site/news`	JWT	Full list incl. drafts + `publishedOnly`.
`POST` / `PUT /{id}` / `PATCH /{id}/publish|unpublish` / `DELETE /{id}`	`/site/news…`	JWT	Manage.

`newsdate` defaults to today on create. Ordered by `newsdate` desc, then `id` desc.

Press

Official press releases — **global or eparchy authored only** (`scope=parish` → 400).

```
// press release (data)
{ "id": 2, "title": "Statement on …", "releaseDate": "2026-05-28", "summary": "…",
  "bodyHtml": "<p>…</p>", "imageUrl": "https://.../p.jpg", "attachmentUrl":
"https://.../p.pdf",
  "scope": "global", "ownerType": "platform", "ownerId": null,
  "isPublished": false, "sortOrder": 0, "createdAt": "…", "modifiedAt": "…" }
```

Method	Path	Auth	Purpose
`GET`	`/site/press/list-published`	public	Published releases. Filters: `scope`, `ownerType`, `ownerId`.
`GET`	`/site/press/published/{id}`	public	One published release.
`GET`	`/site/press`	JWT	Full list incl. drafts + `publishedOnly`.
`POST` / `PUT /{id}` / `PATCH /{id}/publish|unpublish` / `DELETE /{id}`	`/site/press…`	JWT	Manage.

`releaseDate` defaults to today. Ordered by `releaseDate` desc, then `id` desc. **id type:** numeric **Long** for both news and press.

Validation / error notes

- News `headline` / press `title` required → 400.
- News `scope=parish` on an **ERP-client parish** → 400 (use `/manage-news`).
- Press `scope=parish` → 400 (global/eparchy only). Eparchy `ownerId` must exist → 400 otherwise.
- Unknown enum value → 400. Missing id → 404.

Module 11 — Circulars *(Phase 7 — built)*

Near-twin of Pastoral Letters: the circulars list behind the `/circulars` page, **global or eparchy authored only** (`scope=parish` → 400). Adds an official `referenceNo` and a filterable `type`, keyed by `issuedDate`. The page framing (hero / `types[]` / `process[]` / accessNote / cta) is a **Page CMS** row (Phase 2, slug `circulars`) — read it from `GET /site/pages/published/circulars`.

> **Delta from spec:** the spec sketched `/manage-circulars/list-published`. Circulars live at the provider-scope `/site/circulars`.

- **id type:** numeric **Long**.

Resource shape (`data`)

```
{
  "id": 7,
```

```
  "referenceNo": "CIR/2026/001",
  "title": "Revised parish accounting guidelines",
  "type": "administrative",       // open category string; also a list filter
  "issuedDate": "2026-02-10",     // ISO date; defaults to today on create
  "bodyHtml": "<p>…</p>",         // nullable
  "attachmentUrl": "https://.../circular.pdf",  // nullable
  "scope": "eparchy",             // "global" | "eparchy"
  "ownerType": "eparchy",         // "platform" | "eparchy"
  "ownerId": 42,                  // eparchy = template_hierarchy_nodes.id; null
when global
  "isPublished": false,
  "sortOrder": 0,
  "createdAt": "…", "modifiedAt": "…"
}
```

Endpoints

Public (no JWT — gateway-allowlisted):

Method	Path	Purpose
GET	/site/circulars/list-published	Published circulars. Filters: scope, ownerType, ownerId, type.
GET	/site/circulars/published/{id}	One published circular (with bodyHtml). 404 if absent/unpublished.

Management (JWT required):

Method	Path	Purpose
GET	/site/circulars	Full list incl. drafts. Filters + publishedOnly.
POST	/site/circulars	Create.
PUT	/site/circulars/{id}	Update.
PATCH	/site/circulars/{id}/publish | /unpublish	Toggle publish.
DELETE	/site/circulars/{id}	Hard delete.

Listings are ordered by issuedDate desc, then id desc.

Validation / error notes

- title required → 400. scope=parish (or ownerType=parish) → 400 (global/eparchy only).
- Eparchy ownerId must reference an existing template_hierarchy_nodes.id → 400 otherwise.
- Unknown enum value → 400. Missing id → 404.

Module 13 — Photo Gallery *(Phase 9 — built)*

Albums (the publish unit) each containing ordered images. Global / eparchy / **non-ERP parish** — like News, **ERP-client parishes keep galleries in the tenant module** (`/manage-photo-gallery`); a `scope=parish` album whose `ownerId` is an ERP parish → **400**. Images **inherit the album's scope** (no scope fields of their own) and cascade-delete with the album.

> **Delta from spec:** the spec sketched `/manage-gallery-albums/*`. SyroOne gallery lives at the provider-scope `/site/gallery`. Image **writes** are under `/site/gallery/images` (not the album path) so the public image-read path can be allowlisted without exposing a write.

- **id type:** numeric **Long** for albums and images.

Resource shapes (`data`)

```
// album
{ "id": 3, "title": "Feast 2026", "coverImage": "https://.../cover.jpg",
  "scope": "parish", "ownerType": "parish", "ownerId": 11,
  "isPublished": false, "sortOrder": 0, "createdAt": "…", "modifiedAt": "…" }
// image
{ "id": 21, "albumId": 3, "url": "https://.../img1.jpg", "caption": "Procession",
  "sortOrder": 1, "createdAt": "…", "modifiedAt": "…" }
```

Endpoints

Public (no JWT — gateway-allowlisted):

Method	Path	Purpose
`GET`	`/site/gallery/albums/list-published`	Published albums. Filters: `scope`, `ownerType`, `ownerId`.
`GET`	`/site/gallery/albums/{albumId}/images`	Album's images, ordered by `sortOrder`.

Management (JWT required):

Method	Path	Purpose
`GET`	`/site/gallery/albums`	Albums incl. drafts. Filters + `publishedOnly`.
`POST`	`/site/gallery/albums`	Create album.
`PUT`	`/site/gallery/albums/{id}`	Update album.
`PATCH`	`/site/gallery/albums/{id}/publish` | `/unpublish`	Toggle publish.
`DELETE`	`/site/gallery/albums/{id}`	Delete album (cascades to its images).
`POST`	`/site/gallery/images`	Add an image (`albumId` in the body).
`PUT`	`/site/gallery/images/{id}`	Update an image (url / caption / sortOrder).

Method	Path	Purpose
DELETE	/site/gallery/images/{id}	Delete an image.

The public album-images read returns images for the given album id regardless of the album's published state; discovery is gated by albums/list-published (which only returns published albums). Gallery images are non-sensitive public content.

Validation / error notes

- Album title required → 400. Image albumId + url required → 400.
- Album scope=parish on an **ERP-client parish** → 400 (use /manage-photo-gallery).
- Eparchy/parish ownerId must reference an existing node/parish → 400 otherwise.
- Adding/listing images for a missing album → 404. Missing id on mutate → 404.

Module 14 — Services / Mass Timings *(Phase 10 — built)*

Read-only aggregation — no new store. The SyroOne /services page federates the existing parish mass schedules. The **source of truth stays /parish/{parishId}/mass-schedule** (the parish spine); there is no SyroOne mass-timings table and no write surface here. Each parish's mass slots are the existing parish_mass_schedule rows.

> **Delta from spec:** spec path was /services/mass-timings; it lives at /site/services/mass-timings.

Endpoint

Public (no JWT — gateway-allowlisted):

Method	Path	Purpose
GET	/site/services/mass-timings	Federated mass timings, grouped by parish. All params optional.

Query params (all optional):

Param	Type	Notes
eparchy	number (Long)	A template_hierarchy_nodes.id; includes that node **and all descendants** (so passing an eparchy returns all its foranes/parishes).
city	string	Exact, case-insensitive parish city.
day	string	Matches a slot's dayOfWeek (case-insensitive) **or** a DAILY slot. Values: MONDAY… SUNDAY / DAILY / WEEKDAYS.

> With **no** params it returns every active parish that has mass timings — potentially large. The frontend normally passes eparchy and/or city.

Response (data) — grouped by parish

```
[
  {
    "parishId": 11,
    "parishName": "St. Thomas Church",
    "city": "Kochi",
    "hierarchyNodeId": 42,
    "massTimings": [
      { "id": 5, "parishId": 11, "dayOfWeek": "SUNDAY", "massTime": "07:30:00",
        "language": "Malayalam", "notes": null, "sortOrder": 0,
        "createdAt": "…", "modifiedAt": "…" }
    ]
  }
]
```

Parishes are sorted by name; within a parish, slots keep their `sortOrder`. Only parishes that have at least one matching slot are returned. To **edit** mass timings, use the parish spine (`/parish/{parishId}/mass-schedule`) — not this endpoint.

Validation / error notes

- `eparchy` that doesn't reference an existing hierarchy node → 400.

Modules 1–4 — Site singletons (Navigation / Settings / Footer / Home) *(Phase 11 — built)*

The four SyroOne **global** chrome/config blobs. Each is a single record whose `data` is a **free-form JSON object owned by the frontend** (the backend stores and returns it verbatim — same philosophy as the Page CMS section `props`). Global-only: **no `scope`/`owner`, no `clientid`**. **Live-on-save** — no draft/publish workflow (these are site chrome the page always needs). The four keys are seeded as empty `{}` so they always exist.

> **Suggested `data` shapes** (the contract is yours to evolve; the backend doesn't validate them):
>
> - `navigation` → `{ groups: [{ id, label, href?, type: "link"|"dropdown"|"mega", items?, columns? }], contacts: [{ iconKey, text, href }] }`
> - `settings` → `{ siteName, logoUrl, faviconUrl, phone, email, address, socialLinks: [{ platform, url }], copyright }`
> - `footer` → `{ columns: [{ heading, links: [{ label, href }] }], contactBlock, newsletterCopy, legalLinks: [] }`
> - `home` → `{ heroSlides: [{ image, headline, sub, ctaLabel, ctaHref }], stats: [{ value, label }], testimonials: [{ quote, author, role, avatar }], scripture: { verse, reference }, sectionCtas: [] }`

Endpoints

Public (no JWT — gateway-allowlisted, read-only):

Method	Path	Returns (`data`)
`GET`	`/site/navigation`	the navigation object (or `{}` if unset)
`GET`	`/site/settings`	the settings object
`GET`	`/site/footer`	the footer object
`GET`	`/site/home`	the home object

Management (JWT required):

Method	Path	Purpose
`GET`	`/site/singletons`	All four rows incl. timestamps.
`PUT`	`/site/singletons/{key}`	Upsert one singleton; request body **is** the `data` object. `{key}` ∈ `navigation` | `settings` | `footer` | `home`.

Example — set the footer:

```
PUT /site/singletons/footer
Content-Type: application/json

{ "columns": [ { "heading": "Explore", "links": [ { "label": "Directory", "href":
"/directory" } ] } ],
  "newsletterCopy": "Stay updated", "legalLinks": [ { "label": "Privacy", "href":
"/privacy" } ] }
```

Validation / error notes

- `{key}` not in `navigation|settings|footer|home` → 400.
- A `GET` for an unset singleton returns `data: {}` (200), not 404 — so the site degrades gracefully.

Module 18 — Parish Website Content *(Phase 12 — built)*

Per-parish micro-site content for the SyroOne directory (`/directory/[slug]`) — tagline, about, mission, hero slides, gallery images, quick links. **Parish-only, 1:1 with a parish**, and it applies to **all** directory parishes (ERP or not — no ERP guard; this is SyroOne directory chrome, distinct from the tenant Events/News modules). The list fields are free-form JSON owned by the frontend.

> **Lives on the parish spine**, not under `/site/*`: the read is `GET /parish/{parishId}/website-content` (what knorg-syro consumes alongside `/parish/{id}/priests`, `/mass-schedule`, etc.). The **write** is on a distinct path (`PUT /parish/website-content/{parishId}`) so the read path can be gateway-allowlisted without exposing the write.

- **id type:** numeric **Long** (`parishId` is the natural key, 1:1).

Resource shape (`data`)

```
{
  "id": 8,
  "parishId": 11,
  "tagline": "A community of faith",
  "aboutHtml": "<p>…</p>",
  "missionStatement": "…",
  "heroSlides":   [ { "image": "https://.../h1.jpg", "headline": "Welcome" } ],
  "galleryImages":[ { "url": "https://.../g1.jpg", "caption": "Feast" } ],
  "quickLinks":   [ { "label": "Mass Timings", "href": "/directory/st-thomas/mass-timings" } ],
  "createdAt": "…", "modifiedAt": "…"
}
```

`heroSlides` / `galleryImages` / `quickLinks` are stored as JSONB and round-trip verbatim (shapes above are suggestions, not server-validated).

Endpoints

Method	Path	Auth	Purpose
`GET`	`/parish/{parishId}/website-content`	public	The parish's content. If none saved yet, returns an empty shell (`parishId` set, empty lists) — never 404 for an existing parish.
`PUT`	`/parish/website-content/{parishId}`	JWT	Upsert the parish's content (body is the content object).

Validation / error notes

- `parishId` must reference an existing parish → **404** otherwise (both GET and PUT).
- A `GET` for a parish with no saved content returns the empty shell (200), not 404.

Module 17 — Dashboards *(Phase 13 — built)*

Read-only role-scoped aggregations (counters/lists) over the directory spine and the SyroOne content masters. **Not a content store — no new table.** Responses are ad-hoc JSON objects.

> **Delta from spec:** spec paths were `/dashboard/*`; mounted at **`/site/dashboard/*`** to reuse the existing `/site/**` gateway route.

Endpoints

Method	Path	Auth	Purpose
`GET`	`/site/dashboard/global`	public	Platform-wide counters.
`GET`	`/site/dashboard/regional?eparchy={nodeId}`	public	Counters scoped to an eparchy (node + descendants).

Method	Path	Auth	Purpose
GET	/site/dashboard/parish?parishId={id}	public	Counters for one parish.
GET	/site/dashboard/me	JWT	The identified user's view — **provisional** (see note).

Response shapes (data)

```
// global
{ "parishes": 320, "erpParishes": 18, "nonErpParishes": 302, "priests": 540,
"institutions": 75,
  "publishedContent": { "saints": 12, "events": 5, "upcomingEvents": 3,
"announcements": 8,
    "news": 14, "pastoralLetters": 4, "circulars": 6, "press": 2, "galleryAlbums":
9, "feasts": 30 } }

// regional?eparchy=42
{ "eparchyId": 42, "parishesInEparchy": 64,
  "publishedContent": { "announcements": 3, "circulars": 2, "pastoralLetters": 1,
    "events": 4, "upcomingEvents": 2, "news": 5 } }

// parish?parishId=11
{ "parishId": 11,
  "spine": { "priests": 3, "trustees": 5, "massSlots": 7, "ministries": 4,
"announcements": 6 },
  "publishedContent": { "events": 2, "upcomingEvents": 1, "news": 3,
"galleryAlbums": 2 },
  "hasWebsiteContent": true }

// me (provisional)
{ "message": "Personalized dashboard is provisional — pending the SyroOne member-
login method.",
  "userId": "member-abc", "rsvpCount": 2, "rsvps": [ /* the user's SiteEventRsvp
rows */ ] }
```

Notes / errors

- `regional` requires `eparchy` (a `template_hierarchy_nodes.id`) → 400 if missing, 400 if it doesn't exist. `parish` requires an existing `parishId` → 404 otherwise.
- `/me` **is provisional.** It requires an identified user and is JWT-gated (not public). The user id is taken from the gateway `X-User-Id` header (fallback `userId` query param). Until the SyroOne member-login method is defined (same deferral as RSVP), it only echoes the user and lists their RSVPs.