

System Architecture Document

BALM Nexus — Phase 1: Website, CMS & Blog

Client	BALM Consulting Group (BCG)
Prepared by	Wisdom Technologies Inc.
Document status	Draft v0.2 — reflects domain split (balmconsultinggroup.org / app.balmconsultinggroup.org) and confirmed Neon database
Related docs	PRD, Database Schema Design, API Specification (OpenAPI 3.0), Architecture Diagram (.drawio)

Table of Contents

Table of Contents	2
1. Goals & Constraints	3
2. Architecture Style: Modular Monolith, Split Across Two Domains	3
Rationale for the domain split	3
Rationale for the modular monolith (vs. microservices) within each app	3
3. High-Level Architecture	4
4. Component Breakdown	4
5. Key Data Flows	5
6. API Architecture	6
7. Security Architecture	6
8. Deployment Architecture	7
9. Scalability & Caching Strategy	7
10. Monitoring & Observability	8
11. Technology Stack Summary	8
12. Extensibility for Future BALM Nexus Phases	9

1. Goals & Constraints

- Serve a public, SEO-critical website with excellent Core Web Vitals.
- Give non-technical staff a reliable CMS with clear roles (Super Admin, Content Editor).
- Keep hosting cost low and the codebase maintainable by a single future developer if Wisdom Technologies is unavailable.
- Data model must extend cleanly into future BALM Nexus modules (partnership portal, workflow engine, AI scouting, proposal workspace) without a rebuild.

2. Architecture Style: Modular Monolith, Split Across Two Domains

Choice: Two independent Next.js applications sharing a common component/domain-logic package (npm workspace / monorepo) rather than one app with an internal route split:

App	Domain	Purpose
Public Site	balmconsultinggroup.org	SSR/ISR public pages: Home, About, Services, Projects, Team, Contact, Blog. No admin auth surface at all.
Admin CMS + API	app.balmconsultinggroup.org	Authenticated CMS dashboard and the full mutating API (Pages, Projects, Blog, Media, Users, Audit).

Both are backed by a single **Neon (serverless PostgreSQL)** database — **not** microservices, and **not** a single shared app.

Rationale for the domain split

- **SEO isolation.** All domain authority and crawl budget accumulate on **balmconsultinggroup.org**, the only domain that matters for search ranking. The CMS carries zero SEO value and is excluded from indexing entirely (**app.** gets a blanket **noindex** header and a **Disallow: /** in its own robots.txt) without touching the public site's SEO configuration.
- **Cookie/session isolation.** The admin session cookie is scoped as host-only to **app.balmconsultinggroup.org** (never **Domain=.balmconsultinggroup.org**), so it is never sent to the public site's origin. A compromise of a public-facing component (a third-party embed, a form widget) has no session to steal, because no session cookie is ever present there.
- **Independent deploy cadence and blast radius.** The public site is the high-traffic, SEO-critical surface; the CMS is low-traffic and internal. A CMS deploy, migration, or bug cannot take the public site down, and vice versa.

Rationale for the modular monolith (vs. microservices) within each app

- Traffic and team size do not justify microservices overhead (independent deploys, service discovery, distributed tracing) at this stage.

- Clear internal boundaries (services/repositories per domain: Pages, Projects, Blog, Media, Users) inside the Admin/API app give most of the maintainability benefit without the operational cost.
- The module boundaries (see §4) are designed so that a future phase — e.g., the AI opportunity-scouting engine — can be extracted into its own service later if it needs independent scaling, without disrupting this phase.

Considered and rejected for Phase 1: pure microservices (premature complexity), a single app with route-based separation of public/admin (rejected specifically because it cannot give cookie isolation or independent deploys — see above), fully static site with headless CMS SaaS (rejected because BCG wants a **custom**, client-owned CMS rather than a recurring SaaS dependency).

3. High-Level Architecture

See the accompanying architecture diagram (editable [.drawio](#) file, opens in diagrams.net) for the full visual layout. In summary: Visitors reach the CDN in front of [balmconsultinggroup.org](#), which serves the Public Site (Next.js, SSR/ISR); Super Admins and Content Editors authenticate directly against [app.balmconsultinggroup.org](#), which runs the Admin CMS and its API routes. Both the Public Site and the API read/write a single shared Neon (serverless PostgreSQL) database over its pooled connection, and the API additionally talks to Object Storage (media/documents) and fires a revalidation webhook back to the Public Site whenever content is published. Supporting services — Search Console/Analytics, an Email/Newsletter provider, Monitoring (Sentry/uptime), and CI/CD (GitHub Actions) — sit alongside this core flow as shown in the diagram.

Note on the public site's data access: the public site does **not** call the [app](#). API for reads. It queries Neon directly via server-side Prisma calls at request/ISR-revalidation time — one less network hop for a read-heavy, SEO-critical surface, and it removes any reason for the public site to hold API credentials for the admin service. The public site only makes outbound calls for its two write paths: the contact form and the newsletter opt-in, both implemented as its own small serverless route handlers (not routed through [app](#)).

4. Component Breakdown

Component	Domain	Responsibility
Public Site	balmconsultinggroup.org	SSR/ISR rendering of Home, About, Services, Projects, Team, Contact, Blog. Reads content directly from Neon via server-side Prisma — no dependency on the Admin/API app being up. Owns SEO markup, structured data, sitemap/RSS generation, and its own two form-submit routes (contact, newsletter).
Admin CMS	app.balmconsultinggroup.org	Authenticated dashboard for content CRUD, media library, draft/preview/publish, SEO fields, user & role management, audit log viewer. noindex 'd and excluded from crawling.

Component	Domain	Responsibility
API Service	app.balmconsultinggroup.org	Domain services + repository pattern per entity (Pages, Services, Projects, Team, Blog, Media, Users, Audit), colocated with the Admin CMS. Enforces validation and RBAC centrally for every mutating action.
Database (Neon / Postgres)	Shared	System of record — see Database Schema Design doc. Single Neon project; both apps connect via Neon's pooled (PgBouncer) connection string, never the direct connection string, since both run as serverless functions.
Object Storage (GCS/S3)	Shared	Images, project summary PDFs, brochures. Served via signed URLs / CDN, never stored in the DB directly.
Auth	app.balmconsultinggroup.org only	Session-based auth for admin users, scoped as a host-only cookie to app . so it is never sent to the public site's origin. MFA on admin accounts. Public site requires no auth infrastructure at all.
Email/Newsletter	Shared	Transactional email (contact form notifications) + newsletter opt-in provider (e.g., Mailchimp/Resend) via API integration, not built in-house.
Monitoring	Shared	Uptime checks, error tracking (e.g., Sentry), structured logs shipped to a log aggregator.

5. Key Data Flows

Content publish flow: Content Editor edits in Admin CMS (**app**.) → saved as draft in Neon → Super Admin (or self, per role policy) previews → Publish action → API validates permissions → writes to Neon with **published_at** timestamp and audit-log entry → a revalidation webhook is called against the Public Site (**balmconsultinggroup.org**) to trigger ISR revalidation for the affected route, since the two apps are deployed independently and the public site otherwise wouldn't know content changed → CDN serves the updated page on next request.

Media upload flow: Editor uploads file in Media Library (**app**.) → API requests a signed upload URL from object storage → browser uploads directly to storage (keeps large files off the API server) → API stores metadata (filename, URL, type, size) in Neon → asset reusable across Pages/Projects/Blog.

Blog scheduled publish: Editor sets **scheduled_at** → a scheduled job (Vercel Cron on the **app**. project) checks for due posts each minute → flips status to Published in Neon → triggers the same revalidation webhook against the public site, plus sitemap/RSS regeneration.

Public read flow (no `app.` involvement): Visitor requests a page → CDN serves cached ISR output if fresh → on a cache miss/revalidation, the Public Site's own server-side code queries Neon directly via Prisma (pooled connection) → response is cached at the edge again. The Admin/API app is never in this request path, so a CMS outage cannot affect public page loads.

6. API Architecture

- **Location:** the full API lives on app.balmconsultinggroup.org/api/v1/... alongside the Admin CMS. It is the CMS's own backend, not a separately hosted service — see the OpenAPI spec's **servers** block.
- **Style:** REST, versioned under [/api/v1/...](#). GraphQL not warranted at this scale — REST keeps the API layer simple.
- **Representative resources:** [/pages](#), [/services](#), [/projects](#), [/team](#), [/blog/posts](#), [/blog/categories](#), [/media](#), [/users](#), [/auth/session](#), [/audit-logs](#).
- **Public reads bypass this API entirely** (see §5) — the API's **GET** endpoints exist for the Admin CMS's own UI (e.g., listing drafts) and for any future external integration, not for the public site.
- **Public writes** (contact form, newsletter opt-in) are handled by two lightweight route handlers on the Public Site project itself, not on [app](#). — see §3 note.
- **Status codes:** standard REST semantics (200/201/204, 400 validation, 401/403 auth, 404, 409 conflict, 500).
- **Documentation:** OpenAPI 3.0 spec generated from route definitions; published internally for future developer handover.
- **Validation:** schema validation (e.g., Zod) at the API boundary for every mutating endpoint.

7. Security Architecture

- **Authentication:** Credential + hashed password (bcrypt/argon2), optional TOTP-based MFA for all admin accounts.
- **Authorization:** RBAC enforced server-side on every mutating and sensitive-read endpoint — never trust client-side role checks alone.
- **Transport:** HTTPS/TLS everywhere, HSTS enabled.
- **Session scoping:** admin session cookie is host-only on app.balmconsultinggroup.org (**Domain** attribute unset, never wildcarded to .balmconsultinggroup.org), so it is structurally impossible for the public site's origin to read or forward it.
- **Secrets:** Environment-based secret management (Vercel secret manager), including the Neon pooled connection string used by both apps — never committed to source control.
- **OWASP mitigations:**
 - SQL Injection → parameterized queries via ORM (e.g., Prisma).
 - XSS → sanitize/escape all rich-text CMS output on render; restrict allowed HTML in the WYSIWYG editor.
 - CSRF → same-site cookies + CSRF tokens on state-changing admin requests.
 - Broken auth/session → secure, httpOnly, signed session cookies; short-lived tokens; idle timeout.

- Security misconfiguration → locked-down CORS, security headers (CSP, X-Frame-Options), dependency scanning in CI.
- **Rate limiting:** applied to auth endpoints and the public contact form to prevent brute-force and spam abuse.
- **Audit logging:** every publish/edit/delete/role-change recorded with actor, action, target, timestamp.

8. Deployment Architecture

Environment	Purpose	Database
Development	Local + preview deployments per pull request (Vercel preview URLs), one per app	Neon branch created per PR (copy-on-write from a seeded base branch), auto-deleted on PR close
Staging	Mirrors production; used for client review and QA before each phase sign-off	Dedicated long-lived Neon branch
Production	Live public site and CMS	Neon production branch (primary)

- **Frontend:** two separate Vercel projects — one for the Public Site (balmconsultinggroup.org), one for the Admin CMS + API (app.balmconsultinggroup.org) — sharing a common package for types, UI primitives, and Prisma schema via an npm workspace/monorepo.
- **Database:** Neon (serverless PostgreSQL). Both Vercel projects use Neon's pooled connection string (`?pgbouncer=true` for Prisma) since both run as serverless functions; the direct connection string is reserved for migrations only, run from CI, never from a serverless function.
- **CI/CD:** GitHub Actions — lint, type-check, unit/integration tests, dependency vulnerability scan, Prisma migration against a fresh Neon branch, then deploy on merge to `main` (staging) and tagged release (production). Each PR's Neon branch doubles as the database for that PR's Vercel preview deployments on both projects.
- **Infrastructure as Code:** environment configuration captured in version-controlled config (Terraform optional at this scale; at minimum, documented and reproducible environment variables and provisioning steps for both Vercel projects and the Neon project).

9. Scalability & Caching Strategy

- Public pages use **ISR (Incremental Static Regeneration)** so most traffic is served from CDN cache, not the origin — this is the primary scalability lever for a marketing/content site and keeps hosting cost low.

- **Neon scale-to-zero** means the CMS's low, bursty traffic pattern doesn't pay for idle compute; Neon autoscales compute up under load (e.g., a traffic spike from a well-shared blog post) without manual intervention.
- Database indexes on frequently filtered columns (project sector, project funder, blog post status + published_at) — detailed in the Database Schema doc.
- Object storage + CDN for all media, keeping both app tiers stateless and easy to scale horizontally if traffic grows.
- Image optimization via Next.js Image pipeline (responsive sizes, modern formats).

10. Monitoring & Observability

- Uptime monitoring on the public site and admin login.
- Error tracking (e.g., Sentry) across frontend and API.
- Structured logs (JSON) for API requests, especially auth and publish actions, retained for audit purposes.
- Monthly SEO/performance snapshot (Core Web Vitals, indexing status, ranking movement) per the proposal's success-measurement commitment.

11. Technology Stack Summary

Layer	Technology	Rationale
Public site	Next.js (React), balmconsultinggroup.org	SSR/ISR for SEO + performance; reads Neon directly, no admin surface
Admin CMS + API	Next.js (React) + Node.js API routes, app.balmconsultinggroup.org	Shared codebase/language for admin UI and its own backend; isolated cookie/session domain
Database	Neon (serverless PostgreSQL) — confirmed	Pooled connections fit the serverless deploy model on Vercel; per-environment database branching maps directly onto the Dev/Staging/Production split; scale-to-zero keeps cost low for a low/bursty-traffic CMS
File/document storage	Google Cloud Storage or AWS S3	Offloads large binary assets from the DB and app servers
Auth	Session-based RBAC with MFA, host-only cookie scoped to app.	Matches the two-role model (Super Admin/Content Editor) and enforces domain-level session isolation
Hosting	Two Vercel projects (Public Site, Admin CMS + API)	Independent deploys and cache behavior per domain; low ongoing cost, minimal ops burden

12. Extensibility for Future BALM Nexus Phases

The domain services (Projects, Users/Roles, Audit, Media) are designed as bounded modules so that:

- The **Partnership & Collaboration Portal** can reuse the existing Users/Roles and notification infrastructure, adding a new **partnership_inquiries** domain.
- The **Tender/Bid Tracker** and **Proposal Workspace** can reuse the Projects/Funders taxonomy already modeled in Phase 1 rather than redefining sectors and funders from scratch.
- The **AI Opportunity Scouting Engine**, if it needs independent scaling or a different runtime (e.g., Python for ML tooling), can be extracted as a separate service consuming the same PostgreSQL data or an API contract, without touching the public site or CMS.