WDAI Foundation: Tech considerations

Background

We have duct-taped a lot of different tools and automations in WDAI over the years, because we've fallen into our current toolset organically. But duct-taped = hard to streamline. At the very core, we have two systems that have made automations really hard — Wix (website) and Slack (community).

We've thought about updating our website away from Wix. But this will mean cancelling all current membership payments and having individuals sign-up again directly with Stripe.

We're revisiting the move now, because a nonprofit entity = new Stripe (new EIN) anyway, **so it's the right time to make one big move... and think strategically about what we WANT our stack to be.**

Primary goal: Get off Wix (extremely closed system)
Secondary goal: Design stack such that operations can be automated/streamlined

---

Requirements

Commented [1]: @nathan.kupp@gmail.com thoughts on this version?
Assigned to Nate Kupp

Women Defining AI — Community Platform MVP

If we were to build this ourselves

Objectives

1. **Direct Stripe integration** for subscriptions/donations (goodbye Wix Payments).
2. **Agent-ready unified API** (single surface for all agents via MCP).
3. **Gated member portal** for directory, resources, events, and leader tooling.
4. **Reliability & Maintainability**: strong DevOps, observability, and guardrails.

Roles

- **Visitor**: Sees public pages; can purchase/join/donate.
- **Member**: Authenticated; access to member content and directory.
- **Leader**: Authenticated; create/manage community events and resources.
- **Agent**: Scoped access via unified `/api/agents/*` (MCP) endpoints.
- **Admin**: Future (Current - access via github/supabase portal)

Tech Stack

- **Frontend**: Next.js 15 (App Router), TypeScript, Vercel.
- **Auth**: Clerk (roles: visitor, member, leader; service accounts for agents).
- **DB/Storage**: Supabase (Postgres + Row Level Security + Storage).
- **Payments**: Stripe (Checkout + Customer Portal + Webhooks).
- **Video**: Vimeo embeds (gated; no Notion embeds).
- **Events**: Luma API (creation via single backend route).
- **Email**: Mailchimp (audience + tags + automations).
- **Automations/Jobs**:
    - **Vercel Cron** for scheduled jobs.
    - **Durable jobs** via **Inngest** *or* **Supabase Edge Functions + (pg)cron** (choose one; see "Maintainability choices" below).
- **Observability**: Sentry (FE/BE), Vercel Logs, optional OpenTelemetry traces.
- **Queue/Cache/Rate Limit** (optional but recommended): Upstash Redis.
- **CI/CD & QA**: GitHub + Vercel Previews; Jest/Playwright; Zod for runtime validation.
- **API Spec**: Zod → OpenAPI (for MCP tooling + contract tests).

Data Model (minimum viable)

- **users**: id (Clerk), email, name, role, image_url, linkedin_url, visibility (bool), stripe_customer_id, slack_user_id?, mailchimp_id?, created_at/updated_at.
- **memberships**: id, user_id, tier (`monthly | annual | donor_annual`), status (`active | trialing | past_due | canceled`), stripe_subscription_id, current_period_end, cancel_at_period_end (bool).
- **resources**: id, title, type (`video | pdf | link`), url, vimeo_id?, visibility (`member | leader`), created_by, created_at/updated_at.
- **events**: id, luma_event_id, title, starts_at, region, category, created_by, visibility.
- **audit_log**: id, actor (user_id or agent_id), action, target_type, target_id, metadata jsonb, created_at.
- **agent_keys**: id, name, hashed_key, scopes string[], created_at, last_used_at, revoked.

- **integrations**: ids for Slack/Mailchimp/Luma per user, sync timestamps and status fields.
- **Security**: Enable Supabase **RLS** from day one (no public write without server role). Use service role on server-side only; Clerk JWT claims → Postgres policies.

2) Required Automations (and how we'll implement them)

Below, each automation lists: **Trigger** → **Processor** → **Side effects** → **Observability/guardrails**.

A. New member → Mailchimp add + tag "active", start welcome series

- **Trigger**: `checkout.session.completed` (Stripe), `customer.subscription.created|updated` (Stripe).
- **Processor**: `/api/stripe/webhook` verifies signature → upserts **users** + **memberships** → enqueue job `mailchimp_sync_member`.
- **Side effects**:
  1. Mailchimp: upsert subscriber (email, name); apply tags: `member`, `active`, `<tier>`.
  2. Trigger Mailchimp welcome series (via automation ID or tag-based journey).
- **Observability**: Sentry breadcrumb on webhook receipt; idempotency key = Stripe `event.id`; retries with backoff; DLQ for Mailchimp failures; alert on >5 failures/hour.

B. New member → Welcome onboarding series (same job as A)

- **Handled** as part of A; ensure Mailchimp journey starts on tag application.

C. Canceled member → DB updates with last day; auto-remove from member pages at period end

- **Trigger**: `customer.subscription.updated` (Stripe) when `cancel_at_period_end = true` or `status = canceled`.

- **Processor**: Webhook updates **memberships.current_period_end**; enqueue `deactivate_on_period_end` (scheduled run at end-of-day UTC after `current_period_end`).
- **Side effects**: On execution, set membership `status = canceled`, drop "active" tag in Mailchimp, keep "alumni" tag if desired; hide from member directory.
- **Observability**: Guard against clock drift; verify not already reactivated; alert if membership not found or already inactive.

D. Slack user status sync (invite/active/inactive/deactivated) → reflect in core DB

- **Trigger**: **Twice daily Cron** (or on-demand admin action): `/api/jobs/slack-sync`.
- **Processor**: For each member with `visibility=true`, lookup Slack by email → reconcile status to **integrations.slack_status** and **users**. Optionally issue invites (requires Slack admin API/permissions).
- **Side effects**: If deactivated or inactive, mark in DB; optional email nudge for inactive.
- **Observability**: Rate limit handling; partial failures logged & retried; alert on >10% lookup failures; store `last_seen_at`.

Note: Slack "invite" endpoints require elevated scopes (or SCIM on Enterprise). If unavailable, use **shareable invite links** + status checks only.

E. Luma events → 2× daily auto-approval based on membership

- **Trigger**: **Cron** twice daily → `/api/jobs/luma-auto-approve`.
- **Processor**: Fetch pending Luma RSVPs or submissions → by email, verify **active** membership → approve/deny via Luma API.
- **Side effects**: Write audit_log entries; optional Slack DM/Email to requester on decision.
- **Observability**: Alert on Luma API failures; idempotent approvals.

3) Unified Agent Architecture (single surface, MCP-ready)

All agents go through `/api/agents/*` with **scoped keys** (no user cookies). Each key has **narrow scopes** stored in `agent_keys.scopes`. Clerk service users can be used for stronger auditing.

```
/api/agents/
  members    → GET/POST/PATCH/DELETE (Supabase-backed)
  payments   → Stripe ops (read refunds, issue refunds, cancel subs)
  events     → Luma read/create; RSVP approvals (read-only in MVP except
create via leader flow)
  content    → Resources CRUD (Supabase); (Notion read-only if you add it
later for editorial notes—no embeds)
  analytics  → Engagement metrics (emails, events attended, resource views)
```

MCP exposure: publish an **OpenAPI** spec that lists these endpoints + scopes. Agents load the spec via MCP tool manifest.
Guardrails: per-scope rate limits; per-endpoint idempotency; audit every write (actor = `agent:{name}`).

4) DevOps for Maintainability (what to put in place now)

Core DevOps Stack

- **Observability**:
  - **Sentry** (FE + API): release tagging, source maps, alert rules.
  - **Vercel Logs** + optional **Logtail/Better Stack** for longer retention.
  - **Basic health checks**: `/api/health` (checks DB, Stripe, Supabase).
  - **Job dashboards**: Inngest/Trigger.dev UI *or* custom table + admin page.

- **Backgrounds & Scheduling**:
  - **Vercel Cron** for "2× daily" and nightly jobs.
  - **Durable job runner**: pick **one** of:
    - **Inngest** (recommended: robust, replays, UI)
    - **Supabase Edge Functions + (pg)cron** (few deps; Postgres-native schedules)
- **Queues / Idempotency / Rate Limits**: Upstash Redis (lightweight) for webhook idempotency locks, global rate limits, and short job queues.

- **Secrets Management**: Vercel env vars; per-environment (dev/staging/prod); rotate quarterly; principle of least privilege.

- **CI/CD**:
    - GitHub + Vercel Preview Deploys (PRs).
    - GitHub Actions: typecheck, lint, tests, build, **database migration check**.

- **Schema & Migrations**:

    - Use ~~**Drizzle** or~~ **Prisma** for typed schema; migrations run in CI before deploy.
    - Supabase SQL policies (RLS) under version control.

- **Testing**:

    - **Unit** (Jest/Vitest), **Integration** (supertest + local Postgres), **E2E** (Playwright).
    - **Contract tests** for Stripe/Luma/Mailchimp using OpenAPI mocks (MSW) and Stripe test mode.
    - **Webhook signature tests** (invalid signature must fail).

- **SLOs & Alerts** (example baselines):

    - **Checkout success rate** ≥ 98% over 7d.
    - **Webhook processing lag** p95 < 2 minutes.
    - **Leader event creation success** ≥ 99% over 7d.
    - **Cron job success** ≥ 99% last 24h (no consecutive failures).
    - **API error rate** p95 < 1% (5xx).
    - Alert routing: email + Slack #eng-alerts.

- **Backups & Recovery**:
    - Supabase daily backups; quarterly **restore rehearsal** to staging.
    - One-click rollback via Vercel deployments (keep last 10).

- **Runbooks** (in repo /docs/runbooks):
    - Webhook failures (Stripe/Mailchimp/Luma).
    - Job backlog growing.
    - Slack API quota errors.
    - Hotfix flow.

5) Sequence Diagrams (key flows)

Stripe → Member Activation → Mailchimp

```
sequenceDiagram
  participant Stripe
  participant Webhook as /api/stripe/webhook
  participant Jobs as Job Queue
  participant DB as Supabase
  participant MC as Mailchimp
  Stripe->>Webhook: checkout.session.completed (signed)
  Webhook->>DB: upsert user + membership
  Webhook->>Jobs: enqueue mailchimp_sync_member (idempotent)
  Jobs->>MC: upsert subscriber + tags (member, active, <tier>)
  Jobs->>MC: start welcome flow (tag/journey)
  Jobs-->>DB: write audit_log entries
```

Cancel at Period End → Deactivate

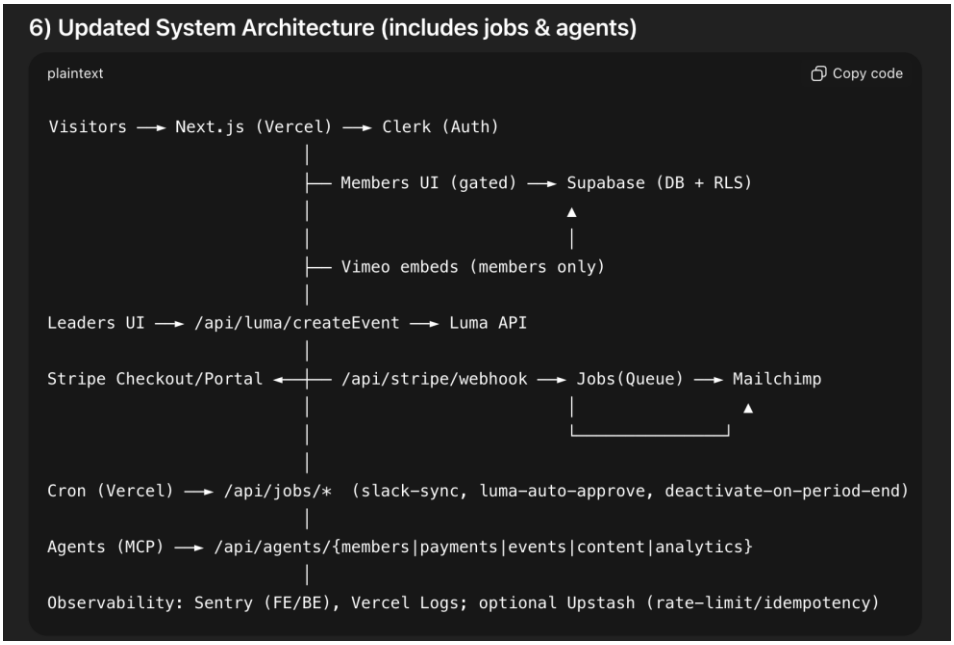
```
sequenceDiagram
  participant Stripe
  participant Webhook
  participant Scheduler as Cron/Scheduler
  participant DB
  Stripe->>Webhook: customer.subscription.updated
  Webhook->>DB: set cancel_at_period_end+current_period_end
  Webhook->>Scheduler: schedule deactivate_on_period_end
  Scheduler->>DB: at period_end -> set status=canceled, hide from
directory
  Scheduler->>MC: remove "active", add "alumni" (optional)
  Scheduler-->>DB: audit_log write
```

Luma Auto-Approval (2× daily)

```
sequenceDiagram
  participant Cron
  participant API as /api/jobs/luma-auto-approve
  participant Luma
  participant DB
  Cron->>API: trigger
  API->>Luma: list pending RSVPs
```

```
API->>DB: check membership status by email
API->>Luma: approve/deny accordingly (idempotent)
API-->>DB: audit_log write; error metrics to Sentry
```

6) Updated System Architecture (includes jobs & agents)



7) Maintainability Review (risks & choices)

- **Webhooks & Jobs are the blast radius** → mitigate with idempotency keys, retries, DLQ, and dashboards.
- **Too many moving parts** → choose **one** durable job system:
  - **Inngest**: best DX + visibility (recommended).
  - **Supabase Edge + (pg)cron**: fewer deps; good if you prefer Postgres-native.
- **Slack invites** can be tricky depending on plan/permissions → build the sync as **status-only** first, add invites later if your workspace allows Admin/SCIM.
- **Notion**: since you banned embeds, keep Notion **out of MVP** UI; if needed for the **Knowledge/Content agent**, add read-only endpoints behind `/api/agents/content`.
- **RLS**: invest early—prevents whole-class bugs.
- **Schema evolution**: version migrations; never hot-edit via console.

- **Single API surface for agents**: reduces maintenance (one auth model, one audit log), but enforce **strict scopes** + rate limits.

8) High-Level Roadmap (2 devs) — with DevOps & Automations

Phase 1 — Foundations (Week 1–2)

- Next.js scaffold, Clerk auth (roles), Supabase schema + **RLS**.
- Stripe Checkout + Customer Portal; `/api/stripe/webhook` with signature verification + idempotency.
- Public pages (static).
- DevOps: Sentry, Vercel envs, CI (lint/type/tests), pick **Inngest** *or* **Supabase Edge+(pg)cron**.
- Tests: unit for auth & webhooks; OpenAPI skeleton.

Deliverables: Running app with auth; Stripe purchase creates user/membership in DB.

Phase 2 — Member Portal (Week 3–4)

- Member profile edit + directory (visibility toggle).
- Vimeo embed pages (courses/recordings) — gated routes.
- Stripe → Mailchimp automation (A+B): job to upsert + tag + start welcome.
- DevOps: alert rules for webhook failures; job dashboard.

Deliverables: Members can log in, see content; new member welcome runs automatically.

Phase 3 — Leader Portal + Luma (Week 5–6)

- Leader dashboard (role-gated), **single form** → `/api/luma/createEvent`.
- Resource admin (CRUD in Supabase) for leaders.
- Luma **2× daily auto-approval** job (E).
- Tests: integration for Luma create/approve; E2E for leader flow.

Deliverables: Leaders can create events; approvals run automatically.

Phase 4 — Cancellation & Slack Sync (Week 7–8)

- Cancel at period end (C): schedule + execution + Mailchimp tag updates.
- Slack sync job (D): status reconciliation; optional invite if permitted.
- Observability: SLO dashboards; error budget alerts; audit_log views.

Deliverables: Members auto-removed on last day; Slack statuses reflected in DB.

Phase 5 — Unified Agent API + MCP (Week 9–10)

- `/api/agents/{members|payments|events|content|analytics}` with **scoped keys**, audit, rate limits.
- OpenAPI spec + MCP tool manifest; contract tests.
- First agents: **Finance** (Stripe reporting), **Support** (refund/cancel), **Engagement** (Mailchimp suggestions draft only).

Deliverables: Agents functional via MCP; all access audited.

Phase 6 — Hardening & Launch (Week 11)

- Load test critical paths; polish UX; content migration.
- Backup/restore rehearsal; runbook sign-off; incident drill.
- Go-live in Vercel Prod.