Technical Architecture & Security Plan

Multi-tenant AI Website Platform (Kuwait)

Next.js + FastAPI + AWS (ECS Fargate, RDS Postgres RLS, Redis, S3)

GOAL	Secure, scalable multi-tenant platform: branded sites + admin dashboard + catalog/projects + orders/donations/pledges + KPIs + safe AI assistant.
DEFAULT DEPLOY	Next.js on Vercel (fast shipping) + FastAPI on AWS ECS Fargate + RDS Postgres (RLS) + Redis + S3 + CloudFront/WAF.
TENANCY MODEL	Shared DB/schema, tenant_id everywhere, Postgres RLS enforced with app.current_tenant set per request.
AI CONTROL	Per-tenant quotas + rate limits + hard caps on tokens/turns; log usage and cost per call.

Prepared for: Dr. Jamal and team | **Date:** February 16, 2026

1. Product Scope (MVP)

- **Multi-tenant websites:** each client has its own subdomain/custom domain, branding, and public pages.
- **Catalog/projects:** products or charity projects with images, descriptions, targets/prices.
- **Capture flows:** orders/donations/pledges + reference numbers + staff notifications.
- **Transparency:** proof/updates (photos, invoices) and status changes (active/funded/completed).
- **KPIs:** traffic-to-outcome tracking via UTMs + conversion metrics.
- **AI assistant:** FAQ + guidance + item matching + safe handoff to staff.

2. Architecture Overview

We separate the system into: (1) web frontend, (2) API backend, (3) background worker, and (4) managed data services. This keeps the MVP simple while staying production-ready.

Layer	Choice	Why
Frontend	Next.js (Vercel default)	Fast deployments, previews, SEO; can move to AWS later if required.
API	FastAPI on ECS Fargate	Long-running service; good for streaming AI responses (SSE) and stabl
Worker	Celery worker on Fargate	Async jobs (emails, media processing, embeddings) without blocking AI
Database	RDS PostgreSQL	Relational core; supports Row Level Security and pgvector when neede
Cache/Rate limits	ElastiCache Redis	Rate limiting, quotas, caching tenant config, AI usage counters.
Storage	S3	Images and proof documents; versioning enabled.
Edge/Security	CloudFront + WAF (API)	TLS, caching, bot protection, OWASP rules, geo/rate policies.

3. Multi-tenancy Model (Critical)

MVP uses a shared database and shared schema. Every tenant-scoped table has a tenant_id. Postgres Row Level Security (RLS) is enabled so the database enforces tenant isolation even if application code has a bug.

3.1 How tenant isolation works

- Request arrives → tenant is resolved from subdomain/custom domain (e.g., *tenant.example.com*).
- API verifies JWT (admin/staff) or treats request as public/guest (donor).
- API sets DB session variable: **SET app.current_tenant = '<tenant_uuid>'**.
- RLS policies restrict all reads/writes to rows where tenant_id matches app.current_tenant.
- S3 objects are stored under a per-tenant prefix (e.g., **{tenant_id}/...**) and pre-signed URLs are issued only after ownership check.

3.2 Tenant data model (minimum)

- **tenants:** slug, name, custom_domain, branding/settings JSON, status, AI budgets.
- **users:** tenant_id, email, role (tenant_admin/staff), identity provider subject (Cognito/Clerk).
- **catalog_items:** tenant_id, product/project fields, Arabic fields, media, status.
- **orders:** tenant_id, type (order/donation/pledge), status, items JSONB, total, payment fields.
- **proofs/updates:** tenant_id, linked to order or catalog item, media, created_by.
- **audit_logs:** append-only admin actions.
- **ai_usage_logs:** tokens, model, estimated cost per call for budget enforcement.

4. Security Controls (Must-have)

4.1 Identity and Access

- Use managed auth: **AWS Cognito** (AWS-native) or **Clerk/Auth0** (fast DX). Do not build auth.
- Roles: platform_admin → tenant_admin → staff. Enforce RBAC on every endpoint.
- Require **MFA** for tenant_admin and staff accounts.
- Short-lived access tokens; rotate refresh tokens.

4.2 Rate limiting and abuse prevention

- Edge limits at WAF/CloudFront (IP-based).
- Application limits in Redis (per-tenant, per-user): login attempts, create-order, chat requests.
- AI hard caps: max input chars, max output tokens, max turns per conversation.

4.3 Data protection and auditing

- RDS encryption at rest + TLS in transit.
- S3 bucket private + versioning enabled; only pre-signed URLs, never public uploads.
- Append-only **audit logs** for all admin writes (who/what/when/IP).
- Separate PII from analytics where possible; mask sensitive fields in logs.

5. Reliability and Operations

5.1 Backups and recovery

- RDS automated backups + point-in-time recovery; manual snapshot before migrations.
- S3 versioning enabled; periodic lifecycle rules.
- Practice restore (monthly) for confidence.

5.2 Observability

- CloudWatch logs and metrics for API and worker.
- Sentry (or equivalent) for exception tracing.
- Basic health checks on ALB; alarms on CPU/memory, DB connections, queue backlog.

5.3 Deployment environments

- **Dev:** docker-compose (local Postgres/Redis) for fast development.
- **Staging:** AWS with smaller instances to test domains, RLS, and payment webhooks.
- **Prod:** AWS with Multi-AZ RDS; separate secrets; controlled releases.

6. AI Assistant Architecture (Safe + Metered)

- **Knowledge sources:** tenant policies, FAQs, catalog/projects, latest proof updates.
- **RAG approach:** start with Postgres (optional pgvector) to retrieve relevant snippets per tenant.
- **Guardrails:** the assistant must not invent amounts/status; only respond from stored data; otherwise escalate to staff.
- **Cost controls:** per-tenant monthly token budget + per-user rate limits + model tiering (cheap model default).
- **Logging:** record tokens/model/cost per call; alert at 80% budget.

7. Cost Model (What actually drives cost)

- **Infrastructure baseline:** ECS tasks, ALB, RDS, Redis, CloudFront/WAF (usually predictable).
- **AI usage:** dominant variable at scale. Must be capped per tenant plan.
- **Storage/bandwidth:** grows with images/videos; control with limits and compression.

Practical rule: price your plans to cover baseline + include an AI quota; sell more AI usage as an add-on.

8. MVP Build Plan (Execution)

Milestone	Deliverables
M1 — Foundations	Repo setup, Docker local dev, tenant resolution, auth, DB migrations, RLS policies.
M2 — Catalog/Projects	CRUD catalog, media upload to S3, public pages + branding.
M3 — Capture + Ops	Orders/donations/pledges, reference numbers, notifications, basic admin inbox.
M4 — Proof + KPIs	Updates/proofs publishing, KPI summary endpoints, event tracking (UTM).
M5 — AI Assistant	Chat endpoint (SSE), RAG retrieval, guardrails, quotas, AI usage logs.
M6 — Hardening	WAF rules, rate limiting, audit logs, backup checks, monitoring alarms.

Note: Payment integration is designed as a plug-in. MVP can launch with pledge + bank transfer/pay-link and later upgrade to KNET/cards via a PSP when clients are ready.