Tech Stack Decisions & Justification

Themes & Motifs — Wedding marketplace/directory for the Philippines

Document purpose	Record the project's core technology choices and the rationale behind them (cost, security, speed of delivery, scalability).
Scope	Web app (Next.js) + data platform (Supabase) + operational integrations (email, payments).
Sources	`contexts_txt_files/ChatGPT Context.txt` and `contexts_txt_files/microsite_context.txt` (plus current repo stack details as implemented)

---

1) Product context (what drives the decisions)

Themes & Motifs is a production-grade wedding marketplace/directory. It is primarily a read-heavy discovery experience (browse/search vendors and promos), with authenticated write flows (reviews, saves, inquiries/leads, vendor onboarding, and admin moderation).

- Core features: vendor listings + filters, vendor detail pages, galleries, reviews/ratings, inquiries, promos/deals.
- Operational needs: verification documents, role-based access control, plan-based feature gating (Standard vs Premium).
- Constraints: keep DevOps minimal while remaining secure and scalable; keep early-stage costs low and predictable.

2) Decision summary

This table is the quick index; each decision is expanded in section 3.

AREA	DECISION	PRIMARY JUSTIFICATION	KEY TRADE-OFFS / NOTES
Frontend framework	Next.js (TypeScript)	SEO-friendly SSR/SSG, App Router, fast UI iteration for a directory-style product	Requires careful caching/data-fetch patterns; server/client boundary must be maintained

AREA	DECISION	PRIMARY JUSTIFICATION	KEY TRADE-OFFS / NOTES
Backend platform	Supabase (Postgres + Auth + Storage)	One managed platform covering DB/auth/files with minimal DevOps; strong security via RLS	Vendor lock-in considerations; service-role usage must be strictly server-only
Authorization	Row Level Security (RLS) in Postgres	Centralized, auditable access rules close to the data; reduces "forgotten checks" risk	Policy complexity grows with features; requires discipline in schema & policy design
File storage	Supabase Storage buckets	Supports galleries/docs without storing blobs in DB; integrates with auth and policies	Need conventions for object keys, privacy levels, and lifecycle management
Email	Supabase email templates; optional Resend	Fast start for transactional templates; upgrade path for deliverability and features	Resend adds another vendor; requires managing secrets and webhooks if used
Payments (planned)	Stripe subscriptions	Standard choice for billing, recurring plans, and webhooks; scales with minimal ops	Requires careful webhook/RLS-safe updates and plan gating logic
ORM / migrations	Prisma (planned)	Clear modeling and migrations for evolving schema; developer productivity	Must align with Supabase-managed schemas/RLS; avoid dual sources of truth
Hosting	Netlify for the web frontend (not yet reflected in repo)	Simple CI/CD for Next.js and low-maintenance hosting; fits "cheap but reliable" goal	Alternative hosts exist; ensure serverless runtime features align with Next.js needs

3) Detailed decisions

3.1 Next.js (TypeScript) for the web app

Decision Build the frontend as a Next.js application using TypeScript.

Rationale:

- **SEO and performance**: vendor directories and detail pages benefit from server rendering and crawlable HTML.
- **Fast iteration**: React component model supports rapid UI changes for search, filters, and dashboards.
- **Maintainability**: TypeScript reduces runtime errors as the schema and roles grow.

Implementation notes (repo-specific):

- The current repo uses Next.js App Router and server components for read-heavy pages.
- Use server-side data fetching for public pages while keeping secret keys server-only.

3.2 Supabase for database, auth, and storage

Decision Use Supabase as the core backend platform: Postgres + Auth + Storage.

Rationale:

- **Minimal DevOps**: one managed platform covers the majority of backend needs.
- **Relational data model**: vendors, categories, promos, reviews, inquiries, saves, and onboarding are naturally relational.
- **Security posture**: Row Level Security can enforce role/ownership rules at the database layer.
- **File handling**: image galleries and verification documents work well with object storage.

Cost / budget notes (early stage):

- Supabase free tier to start, with a predictable upgrade path (noted target: roughly $25/mo early stage).

3.3 Row Level Security (RLS) as the default authorization strategy

Decision Use Postgres RLS policies (with helper functions) to protect user-facing tables.

Rationale:

- **Correctness**: prevents accidental data leaks if an endpoint or query is added without proper checks.
- **Multi-role support**: maps cleanly to soon-to-wed vs supplier vs admin access patterns.
- **Scales with features**: supports future plan gating and premium-only visibility rules at the data layer.

Security & implementation principles:

- Never trust the frontend for permissions.
- Separate public/private data where necessary.
- Use foreign keys and indexes on frequently filtered columns.

3.4 Supabase Storage buckets for images and documents

Decision Store images/documents in buckets (e.g., vendor galleries, profile photos, verification docs, promo images).

Rationale:

- **Performance and cost**: object storage is the right place for files; avoid putting blobs in Postgres.
- **Security**: integrate access patterns with auth and policies.
- **Operational simplicity**: one platform for both data and files.

Key operational note: establish a stable convention for object keys (vendor id, user id, document type) and public/private buckets.

3.5 Email: templates now, provider upgrade path later

Decision Start with Supabase email templates; optionally integrate Resend for improved deliverability and flexibility.

Rationale:

- Transactional emails are needed for onboarding and workflows (welcome, verification, inquiry, approval).
- Start simple, then upgrade provider capabilities as volume grows.

3.6 Payments (planned): Stripe subscriptions for vendor plans

Decision Use Stripe for subscription plans (Standard/Premium) when billing is implemented.

Rationale:

- Supports recurring billing, trials, coupons, and robust webhook events.
- Scales without custom payment infrastructure.

Implementation notes: ensure webhook processing updates subscription state safely (server-only secrets; RLS-aware writes via service role or trusted server logic).

3.7 ORM/migrations (planned): Prisma

Decision Use Prisma for schema modeling and migrations.

Rationale:

- The schema is non-trivial and will evolve (vendors, promos, reviews, inquiries, onboarding, fairs, analytics).
- Migration tooling supports safe iteration and repeatable deployments.

Important alignment constraint: RLS policies, SQL functions/triggers, and Supabase-managed schemas must remain compatible and should not be accidentally overwritten by ORM tooling.

3.8 Hosting and cost target

Decision Host the frontend on Netlify (as stated in the context), with Supabase as managed backend.

Rationale:

- Matches the goal of a cheap but reliable architecture with minimal ops overhead.
- Provides a straightforward upgrade path as traffic and features grow.

Early-stage budget expectation (from context): approximately $40–60/month combined (hosting + Supabase + email provider), excluding Stripe transaction fees.

4) Data model & feature implications (why Postgres-first works)

The feature set implies a relational core: vendor directory, many-to-many taxonomy (categories, affiliations/badges), reviews tied to users and vendors, saves (composite uniqueness), inquiries with status, onboarding/verification workflows, and fair registrations.

- **Search & filters**: start with indexed relational queries; add a specialized search engine only if needed later.
- **Analytics**: store counts/events (views/saves/inquiries/clicks) as tables or rollups; denormalize only when necessary.
- **Plan gating**: model plans/subscriptions and enforce visibility rules using a mix of RLS and server-side checks.

5) Operational practices and guardrails

- **Schema integrity**: use foreign keys; keep timestamps (`created_at`, `updated_at`) on core tables.
- **Performance**: add indexes on frequently filtered fields (category joins, location, rating, active/featured flags).
- **Soft deletes**: prefer `deleted_at` where removal should be reversible/auditable.
- **Secrets**: never expose service role keys to the browser; keep admin controls behind server-only checks.
- **Security defaults**: treat RLS as mandatory for user-facing tables; don't rely on UI-only restrictions.

6) Roadmap alignment (build order)

1. Authentication & role assignment
2. Core CRUD workflows (vendors, reviews, inquiries, saves)
3. File uploads to Storage buckets
4. User-facing pages + vendor dashboard + admin dashboard
5. Email triggers for key workflows
6. Stripe subscriptions (Premium plans)
7. Testing (end-to-end, RLS enforcement, UI validation)
8. Deployment (frontend hosting + Supabase)

Document: `docs/TECH_STACK_DECISIONS.html`