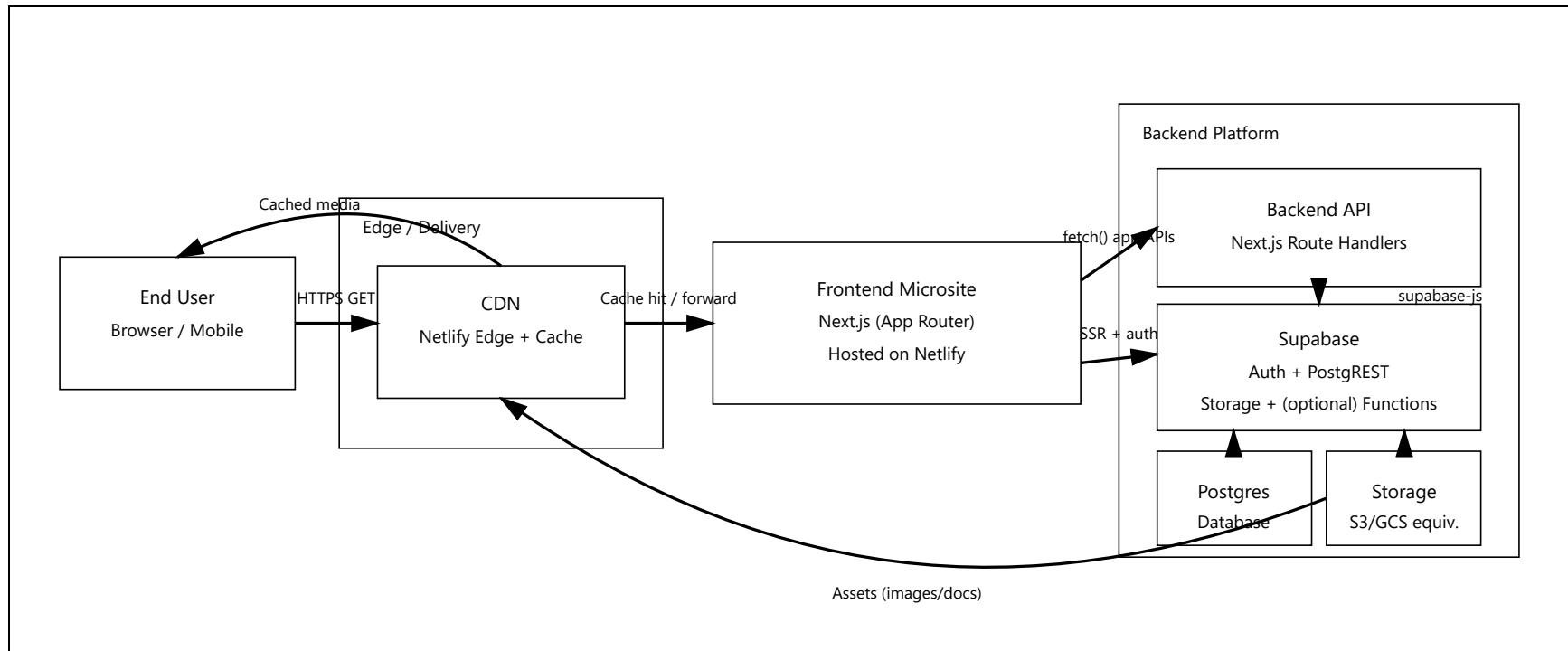# System Architecture Diagram

Themes & Motifs — frontend microsite, backend API, database, object storage (S3/GCS equivalent), and CDN

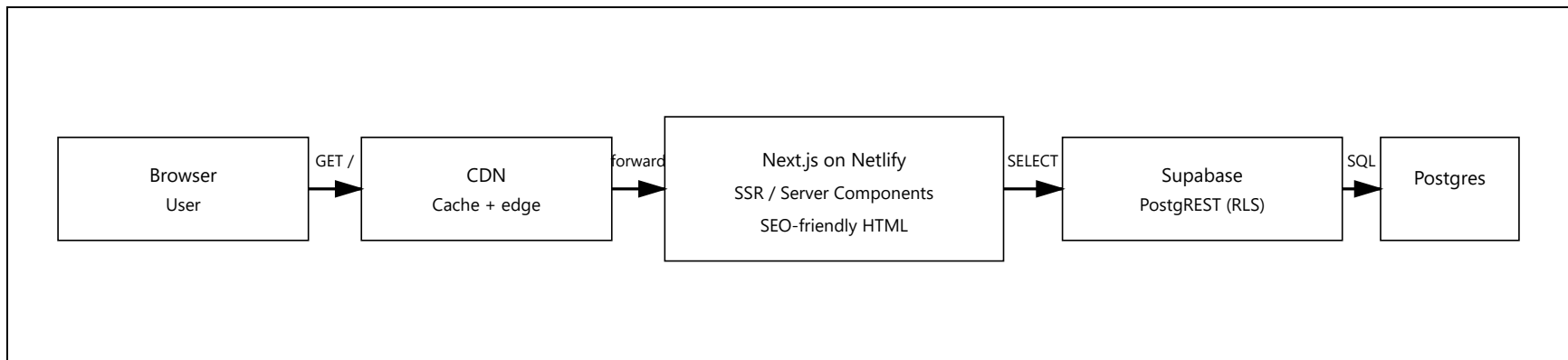## 1) System Context (high level)

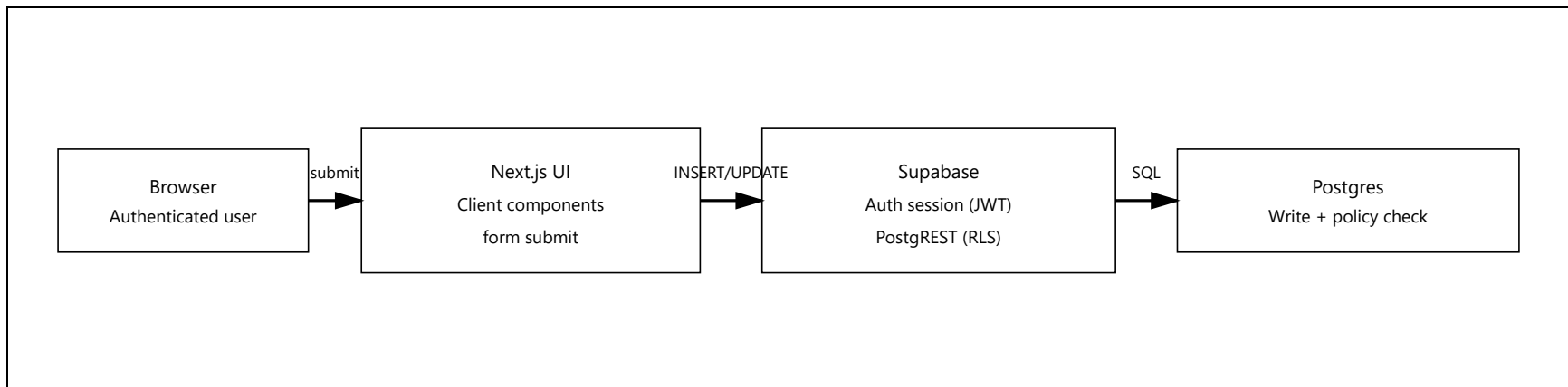| ☐ Request to **Next.js Route Handlers** (app-owned API) | ☐ Direct **Supabase** usage for SSR/auth/data access | ☐ **Storage** is object storage (S3/GCS equivalent); typically delivered via CDN |

2) Key Request & Data Flows (simplified)

2.1 Public browse/search (read-heavy)

Browser
User

GET /

CDN
Cache + edge

forward

Next.js on Netlify
SSR / Server Components
SEO-friendly HTML

SELECT

Supabase
PostgREST (RLS)

SQL

Postgres

2.2 Authenticated writes (reviews, saves, inquiries)

Browser
Authenticated user

submit

Next.js UI
Client components
form submit

INSERT/UPDATE

Supabase
Auth session (JWT)
PostgREST (RLS)

SQL

Postgres
Write + policy check

2.3 Media upload & delivery (images/docs)

Browser
Uploader

upload

Supabase
Auth + Storage API
bucket/prefix policies

PUT object

Object Storage
Supabase Buckets

GET media

CDN
Caches media
delivers globally

cached bytes (images/docs)
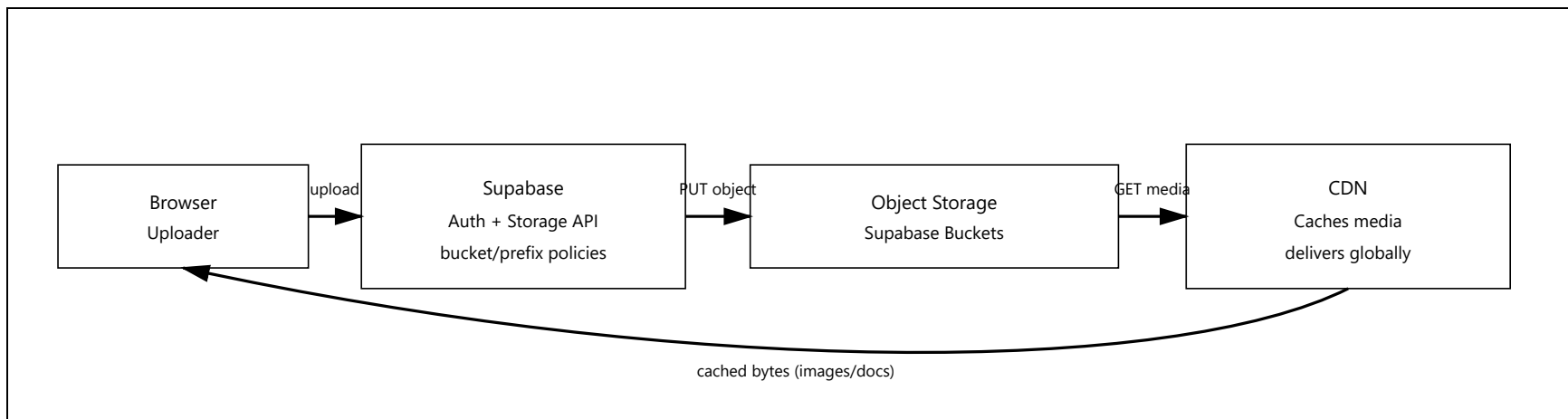
3) Component Notes

- **Frontend microsite (Next.js on Netlify)**

  - Renders the public directory experience (vendors, promos, categories, detail pages).
  - Uses Server Components/SSR for SEO-friendly pages and fast first paint.
  - Uses Route Handlers for app-owned endpoints (e.g., filtered vendor search, admin patch endpoints).

- **Backend API**

  - Implemented primarily as **Next.js Route Handlers** under `src/app/api/**`.
  - Calls Supabase using either:

    - **Anon key** (public reads constrained by RLS), or
    - **Service role key** (admin-only actions; bypasses RLS and must remain server-only).

- **Supabase (backend platform)**

  - **Auth** provides login/session management and role-based access.
  - **PostgREST** provides table access with RLS and policies as the primary security layer.
  - **Storage** provides buckets for images and documents (functionally equivalent to S3/GCS object storage).
  - **(Optional) Edge Functions** can be used later for workflows needing secrets or non-PostgREST logic (e.g., Stripe webhooks, email triggers).

- **Database (Postgres)**

  - Primary system of record for vendors, promos, taxonomy, users/roles, reviews, inquiries, saves, and onboarding/verification workflows.
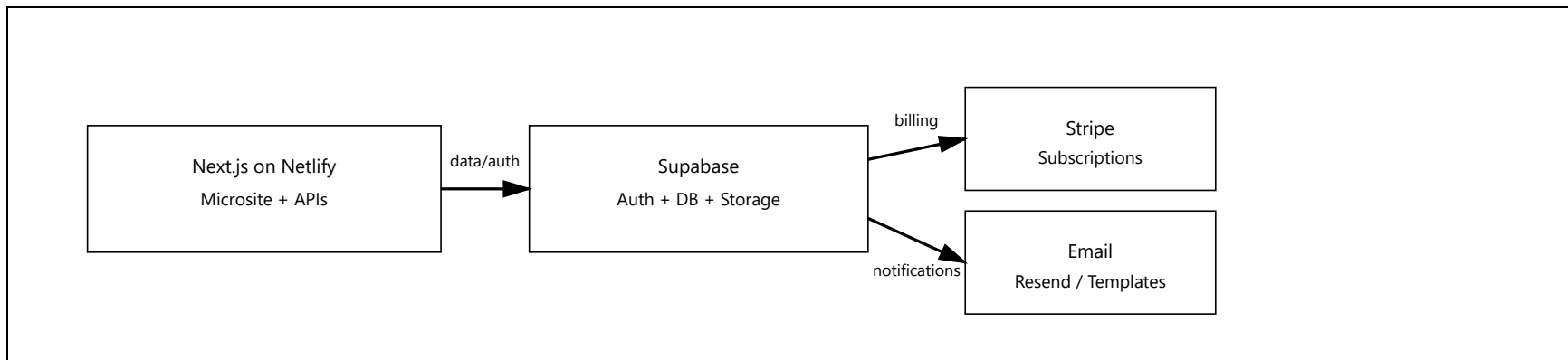
- **Object storage (S3/GCS equivalent)**

  - Stores binary assets (vendor galleries, promo images, profile photos, verification documents).
  - Policies enforce bucket-level and path/prefix constraints (e.g., `<vendor_id>/...`).

- **CDN**

  - Serves cached HTML/JS/CSS for the microsite and cached media assets for faster global delivery.

---

4) Optional External Integrations (planned)

```
┌─────────────────────────────────────────────────────────────────────────────────────────┐
│                                                                    ┌──────────────────┐  │
│                                                          billing   │      Stripe       │  │
│                                                          ────────▶ │   Subscriptions   │  │
│  ┌──────────────────┐    data/auth   ┌──────────────────┐          └──────────────────┘  │
│  │ Next.js on Netlify│ ─────────────▶ │    Supabase      │                                 │
│  │  Microsite + APIs │                │ Auth + DB + Storage│         ┌──────────────────┐  │
│  └──────────────────┘                └──────────────────┘          │       Email       │  │
│                                               notifications ─────▶  │ Resend / Templates│  │
│                                                                     └──────────────────┘  │
└─────────────────────────────────────────────────────────────────────────────────────────┘
```

These integrations are optional and can be added later via Supabase Edge Functions or server-side endpoints.

Generated as static HTML with inline SVG diagrams (no Mermaid dependency).