Bidion MVP System Design Specification

Theory-First Internal Draft & Design Rationale

Jason Ramirez

Draft v0.1 — September 20, 2025

Intended audience: Bidion engineering (backend, frontend, DevOps), product management, and founding leadership.
See §2 for the disclaimer.

Contents

1 Definitions

Table 1: Definitions of Terms and Acronyms

Term	Definition
API	Application Programming Interface; our backend HTTP surface that the frontend calls.
AuthN / AuthZ	Authentication (verifying identity) and Authorization (verifying permissions).
BullMQ (`BullMQ`)	A Node.js job/queue library backed by Redis for background processing (e.g., parse → embed → score chains).
CDN	Content Delivery Network; serves static assets (JS/CSS/images) close to users.
Chunk / Chunking	Splitting long documents into bounded text segments to embed and search efficiently.
Cosine Similarity	Measure of similarity between vectors using the cosine of the angle between them; used to compare embeddings.
Embedding	Dense vector representation of text produced by an ML model; enables semantic similarity search.
HNSW / IVF / IVFFLAT	Approximate nearest-neighbor index types used by `pgvector` (*Hierarchical Navigable Small World / Inverted File*; IVFFLAT is a specific IVF variant).
JWT	JSON Web Token; compact token used for authenticated API requests.
KB	Knowledge Base; internal documents (product sheets, past proposals) used for grounding responses.
K8s	Kubernetes; container orchestration platform.
KNN	k-Nearest Neighbors; returns the k most similar items (e.g., vectors) to a query.
LLM	Large Language Model (e.g., OpenAI GPT-class models).
MVP	Minimum Viable Product; smallest feature set that delivers user value for validation.
NestJS / Express	Node.js web frameworks used to implement the API layer.
Next.js	React framework used for the frontend (App Router).
OCR	Optical Character Recognition; extracting text from scanned PDFs/images (deferred for MVP).
OpenAI API	OpenAI services used for embeddings and chat completions.
Offering	A catalog entry describing the business capability/product being matched against RFPs.

Term	Definition
pgvector (`pgvector`)	PostgreSQL extension adding a `vector` type and similarity operators for embeddings.
PII	Personally Identifiable Information; sensitive data requiring careful handling.
PostgreSQL (Supabase)	Managed relational database used for metadata and `pgvector` vector search.
RAG	Retrieval-Augmented Generation; generate text using LLMs grounded by retrieved context chunks.
RFP	Request for Proposal; procurement document with requirements to which offerings are matched.
Redis	In-memory data store; used here as the queue backend for `BullMQ`.
S3 / Supabase Storage	Object storage for uploaded files; S3-compatible API provided by Supabase Storage.
Semantic Search	Similarity search over embeddings to find conceptually related text, not just lexical matches.
Supabase	Managed platform providing Postgres (with `pgvector`), Auth, and Storage used in this MVP.
Vector Search	Querying a vector index (e.g., `pgvector`) to find nearest embeddings by cosine (or other) similarity.

2 Introduction

Purpose. This document is an internal, *theory-first* design plan for the Bidion MVP. It consolidates architectural intent, key concepts, and proposed strategies/procedures to guide the initial implementation and facilitate structured review. It is not a production runbook.

Status / Disclaimer. The contents are non-final and subject to change as we prototype, test, and receive pilot feedback. Configuration values (for example, pool sizes, SLOs, and index settings) are illustrative targets to be validated under load; they do not constitute commitments to specific vendors, capacities, or dates.

Scope. This draft covers the core system shape and interfaces: Next.js frontend, NestJS API, BullMQ workers, and PostgreSQL+`pgvector` with RLS; plus ingestion, matching, retrieval, and observability touchpoints. Detailed production runbooks, security audits, and multi-region data strategies are *out of scope* for this revision.

Audience. Bidion engineering and adjacent stakeholders (product, infrastructure). The goal is to provide sufficient specificity to start implementation spikes, create RFCs, and make early

trade-offs—while avoiding premature lock-in. (Treat statements herein as informational and evolving rather than binding roadmaps.)

How to use this doc.

- Treat numbers as *initial targets*; refine via load tests (k6), prototypes, and early pilot feedback.
- Use sections marked "MVP-first" to prioritize; items labeled v1+ are intentionally deferred.
- Log deviations (trade-offs, vendor changes) as short RFCs; update this spec incrementally.

Validation plan (high level). Prove the design through: (1) unit/integration tests, (2) short k6 PR runs plus nightly soaks, (3) pilot-org trials with telemetry, and (4) cost/throughput reviews. Findings will update thresholds, capacities, and feature flags before GA.

Change policy. This document is versioned; assumptions, limits, and diagrams may change as we learn. When implementation diverges materially, update this doc or attach a brief RFC link.

3 Project Objective & Assumptions

Goal (MVP). Let a business ingest its offerings/capabilities, ingest RFPs from public/commercial sources, and match them with a clear score & rationale. Provide a dashboard to review matches, collaborate, and draft responses using OpenAI grounded on an internal knowledge base.

Delivery & Operational

- Single tenant at first; design for multi-tenant later.
- Team size 2–4 devs; MVP timeline $\geq$ *6 months* (alpha $\sim$ 3 months, beta $\sim$ 4–5 months, GA $\geq$ 6 months).
- Public RFP sources via simple scrapers/APIs (phase 1: CSV upload/manual).
- Internal KB = PDFs/Docs uploaded by users.
- Budget-conscious; prefer managed services or light VM/K8s.

Workload Assumptions

General

- Traffic is *not* evenly distributed across time or tenants (orgs). Weekdays 9–6 local time are peak; a few heavy users dominate load.
- Creating or uploading an RFP should be fast (P95 $\leq$ 500 ms for metadata create; streaming upload for files).
- Recomputing matches is near real-time for typical docs; queued for very large PDFs or bulk backfills.
- **Active tenants/users (MVP pilot):** *50–200 orgs*; *10k–25k* monthly active users (MAU).
- **Content volume (monthly, MVP):**
  - 6,000–10,000 new RFPs (200–330/day; end-of-quarter spikes $\times$2–3).

  - 40,000–80,000 offering updates (catalog edits, new offerings).
  - 120,000–250,000 KB document ingests/updates (policies, past proposals, appendices).

Timeline / Reads vs Writes

- Viewing RFPs, matches, and dashboards should be fast (cacheable reads; P95 $\leq$ 500 ms).
- The system is **more read-heavy than write-heavy**: reads (match lists, offering lookups, KB previews) dominate UI traffic.
- Ingest paths (RFP upload $\rightarrow$ parse $\rightarrow$ embed) are **write-heavy bursts** and run via workers with backpressure.

Search & Retrieval (RAG)

- Searching should be fast (vector KNN P95 $\leq$ 200 ms under nominal load).
- Workload is read-heavy (assistant questions, filters, keyword+vector hybrid).
- Typical assistant flow: shortlist m=200 via vector search $\rightarrow$ re-rank with domain scoring $\rightarrow$ return top k=50.

Back-of-the-Envelope Usage (MVP)

- **Monthly reads (API GETs)**: *25–45 million*/month (includes CDN edge hits).
  Reasoning: 10k–25k MAU $\times$ $\sim$40–60 reads/day $\times$ 30 days $\approx$ 12M–45M.
- **Monthly writes (API POST/PUT/PATCH)**: *1–3 million* (UI edits, comments, status changes).
- **Monthly assistant queries (vector searches)**: *0.5–2 million.*
- **Monthly match recomputes**: 6k–10k RFPs $\times$ (initial compute + $\sim$2 updates) $\approx$ *18k–30k* recomputes.

Size per Entity (storage planning)

- **RFP (median)**: 40 pages, $\sim$12k words $\Rightarrow$ $\sim$50 chunks (avg 240 words/chunk).
- **Embedding vector (3072-d float32)**: 3072 $\times$ 4 bytes = **12 KB** per vector.
- **Embeddings per RFP**: 50 chunks $\times$ 12 KB $\approx$ **600 KB** (vectors only).
- **Text/PDF storage (RFP)**: median **2–5 MB** per file (depends on scans/figures).
- **Offering record**: $\sim$2 KB metadata; embedding $\sim$12 KB.
- **KB doc (median)**: 10 pages $\Rightarrow$ $\sim$12 chunks $\Rightarrow$ $\sim$144 KB vectors.

Monthly Storage Growth (vectors only, MVP)

- RFP vectors: 6k–10k RFPs × 600 KB ≈ **3.6–6 GB/month**.
- KB vectors: 120k–250k docs × 144 KB ≈ **17–36 GB/month**.
- Offering vectors (updates, net new): 40k–80k × 12 KB ≈ **0.5–1 GB/month**.
- **Total vectors/month ≈ 21–43 GB ⇒ 0.25–0.5 TB/year**.

Throughput (from monthly to per-second, MVP)

Use the same conversion heuristic; note that bursts can be 5–10× the averages and are absorbed via CDN and queues.

- **Read QPS (avg)**: *10–20 requests/s* (25–45M reads/month). *Peak:* 5–10×.
- **Assistant search QPS (avg)**: *0.2–0.8 requests/s* (0.5–2M/month). *Peak:* 5×.
- **Writes QPS (avg)**: *0.5–1.5 requests/s* (1–3M/month). *Peak:* 5×.
- **Match recompute rate**: *12–20/min* at steady state; bursty and handled by workers.

Handy Conversion

- ≈ 2.5 million seconds/month.
- 1 request/second ≈ 2.5 million requests/month.
- 40 requests/second ≈ 100 million requests/month.
- 400 requests/second ≈ 1 billion requests/month.

Performance-Critical Operations (Bid/Proposal Context)

- **RFP create/upload**: immediate user feedback; background parsing/embedding starts within < 5 s.
- **Matches fetch**: cached and paginated; P95 ≤ 500 ms.
- **Assistant ask**: shortlist + re-rank ⇒ P95 KNN ≤ 200 ms, end-to-end answer target < 2 s.
- **Bulk ops**: throttled and queued (embed, re-score), with progress UI.
- **Viewing RFP matches:** *p95* $\leq$ *500 ms* for `GET /rfps/:id/matches` (precomputed).
- **Assistant draft (first token):** *TTFT* $\leq$ *1.5 s*, streaming thereafter.
- **Full-text search across RFPs/KB:** *p95* $\leq$ *300 ms* for query ≤ 3 terms.
- **Attachment preview (PDF/doc first page):** *p95* $\leq$ *700 ms* with CDN caching.
- **RFP create/update (form submit):** server time $\leq$ *250 ms*; heavy work async via queue.
- **Enqueue parse/embed jobs:** $\leq$ *150 ms* API acknowledgement; visible queue status in $\leq$ *1 s*.
- **Parse → matches availability:** *P50* $\leq$ *30 s*, *P95* $\leq$ *90 s* for 10–20 page PDFs.
- **Offerings lookup for mapping to RFPs:** *p95* $\leq$ *400 ms*.

- **Dashboard load (recent RFPs, tasks):** *Largest Contentful Paint* $\leq$ *2.5 s*; API backing calls *p95* $\leq$ *400 ms*.
- **Export (CSV/Docx) kickoff:** $\leq$ *200 ms* to acknowledge; download ready notification *P95* $\leq$ *60 s*.
- **Auth/session resume:** $\leq$ *150 ms* token refresh; cold login $\leq$ *600 ms* server time.

4 MVP Scope

4.1 In Scope (MVP Deliverables)

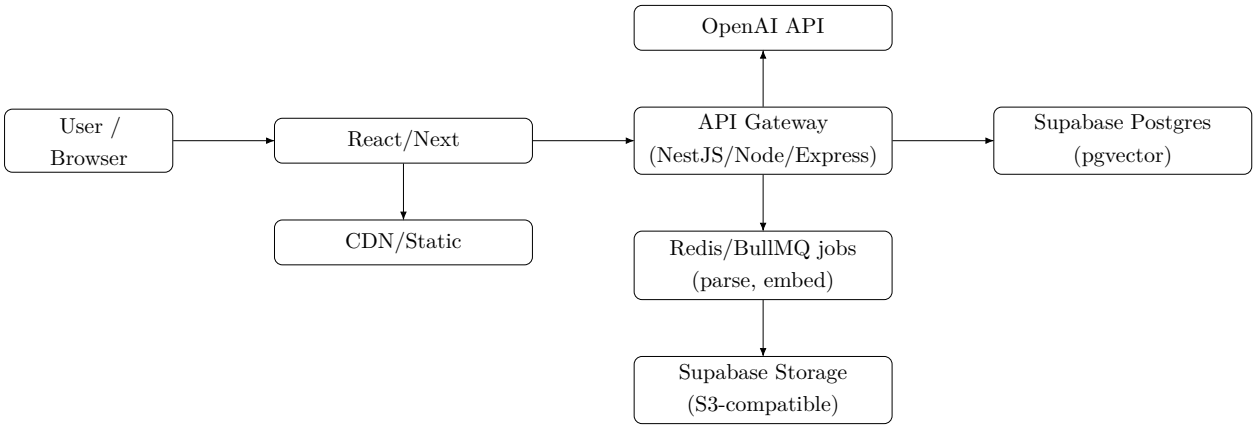
- AuthN/AuthZ, org/user accounts
- Data ingestion: Offerings (forms + CSV), RFPs (URL/PDF/manual), KB uploads
- Matching engine v1: normalization $\rightarrow$ embeddings $\rightarrow$ vector search; hybrid score
- Match review UI: score, snippet highlights, rationale
- Draft response assistant (RAG over KB + RFP)
- Dashboard & basic workflow statuses
- Background jobs (parsing, embeddings, scoring)
- Basic notifications (email/Slack webhook)

4.2 Out of Scope (Deferred / Post-MVP Backlog)

- Fine-grained RBAC + billing
- Advanced procurement integrations; redlining
- Learning-to-rank/ML re-ranker

5 High-Level Architecture

- **Frontend:** Next.js (App Router).
- **Backend:** NestJS.
- **DB:** Supabase PostgreSQL + `pgvector`.
- **Storage:** S3/Supabase Storage.
- **Jobs:** Redis + `BullMQ`.
- **Auth:** Hosted (e.g., Supabase/Auth0/Clerk).
- **LLM:** OpenAI embeddings (text-embedding-3-large), GPT-4 class for RAG.



Data flow:

1. Ingest RFPs/offerings/KB → *parse* → *embed* → *score*.
2. Store embeddings in `pgvector`; raw files in S3; metadata in Postgres.
3. UI queries API for matches and drafts; assistant uses RAG over top chunks.

6 Data Model (v1)

6.1 Table Specifications

Table 2: Table: `orgs`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
name	text		
created_at	timestamptz		default `now()` (if present)

Table 3: Table: `users`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
email	text	UQ	unique, lowercased
name	text		
role	text		enum: `admin|member`
org_id	uuid	FK	→ `orgs.id`
created_at	timestamptz		default `now()`

6.2 Database Schema & Indexes (Supabase Postgres + `pgvector`)

Listing 1: Core tables, vector storage, and indexes

Table 4: Table: `documents`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
kind	text		`rfp|kb|offering` (CHECK)
ref_id	uuid		optional link (e.g., offering id)
filename	text		
storage_url	text		Supabase Storage signed URL or path
mime	text		e.g., `application/pdf`
created_at	timestamptz		default `now()`
status	text		optional: `parsed|embedded|scored`

Table 5: Table: `chunks`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
document_id	uuid	FK	→ `documents.id`, `ON DELETE CASCADE`
idx	int		chunk order
section	text		optional (e.g., "Requirements")
text	text		normalized chunk content

Table 6: Table: `offerings`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
title	text		
description	text		
tags	text[]		GIN index recommended
created_at	timestamptz		default `now()`
updated_at	timestamptz		default `now()`

```
-- Supabase: enable required extensions in our project
CREATE EXTENSION IF NOT EXISTS pgcrypto;
CREATE EXTENSION IF NOT EXISTS vector;

-- documents & chunks
CREATE TABLE IF NOT EXISTS documents (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  kind        text NOT NULL CHECK (kind IN ('rfp','kb','offering')),
```

Table 7: Table: `embeddings`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
kind	text		`rfp|kb|offering` (CHECK)
ref_id	uuid		document or entity id
chunk_id	uuid		nullable for offering-wide vector
vector	vector(3072)		pgvector (cosine ops); IVF-FLAT/HNSW index
text	text		source text for provenance
meta	jsonb		optional

Table 8: Table: `matches`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
rfp_id	uuid	FK	→ `documents.id` (`kind='rfp'`)
offering_id	uuid	FK	→ `offerings.id`
score	double precision		final hybrid score $[0, 1]$
reasons	jsonb		snippets/keywords/penalties
created_at	timestamptz		default `now()`

Table 9: Table: `score_config`

Column	Type	Key	Notes
org_id	uuid	PK/FK	→ `orgs.id` (1:1)
boosts	jsonb		`{"soc2":0.25,"24/7":0.15}`
required	jsonb		`["soc2","24/7"]`
forbidden	jsonb		`["on-prem only"]`

Table 10: Table: `work_items`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
rfp_id	uuid	FK	→ `documents.id` (`kind='rfp'`)
title	text		
status	text		`new|in_progress|review|done`
assignee_user_id	uuid	FK	→ `users.id` (nullable)
created_at	timestamptz		default `now()`

```
10  ref_id      uuid,                -- optional foreign link (e.g., offering id)
11  filename    text,
```

Table 11: Table: `audit_logs`

Column	Type	Key	Notes
id	uuid	PK	`gen_random_uuid()`
org_id	uuid	FK	→ `orgs.id`
actor_user_id	uuid	FK	→ `users.id`
action	text		e.g., `create_rfp`, `match_recompute`
subject_kind	text		`document|offering|match|...`
subject_id	uuid		referenced entity id
meta	jsonb		optional context
created_at	timestamptz		default `now()`

```
  storage_url text,
  mime        text,
  created_at  timestamptz NOT NULL DEFAULT now()
);

CREATE TABLE IF NOT EXISTS chunks (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  document_id uuid NOT NULL REFERENCES documents(id) ON DELETE CASCADE,
  idx         int  NOT NULL,        -- chunk order
  section     text,                 -- optional 'Requirements', etc.
  text        text NOT NULL
);

-- embeddings (pgvector)
CREATE TABLE IF NOT EXISTS embeddings (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  kind        text NOT NULL CHECK (kind IN ('rfp','kb','offering')),
  ref_id      uuid NOT NULL,           -- document_id (or offering id if we embed inline)
  chunk_id    uuid,                    -- null for offering-wide vectors if we store one
  vector      vector(3072) NOT NULL,
  text        text NOT NULL,
  meta        jsonb DEFAULT '{}'::jsonb
);

-- matches
CREATE TABLE IF NOT EXISTS matches (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  rfp_id      uuid NOT NULL,           -- documents.id where kind='rfp'
  offering_id uuid NOT NULL,           -- documents.id where kind='offering'
```

```
  score       double precision NOT NULL,
  reasons     jsonb NOT NULL DEFAULT '{}'::jsonb,
  created_at  timestamptz NOT NULL DEFAULT now()
);

-- keyword/rules (simple MVP)
CREATE TABLE IF NOT EXISTS score_config (
  org_id      uuid PRIMARY KEY,
  boosts      jsonb NOT NULL DEFAULT '{}'::jsonb,  -- {"soc2":0.25, "24/7":0.15}
  required    jsonb NOT NULL DEFAULT '[]'::jsonb,  -- ["soc2","24/7"]
  forbidden   jsonb NOT NULL DEFAULT '[]'::jsonb   -- ["on-prem only"]
);

-- offering catalog (you can also model as documents(kind='offering'))
CREATE TABLE IF NOT EXISTS offerings (
  id          uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id      uuid NOT NULL,
  title       text NOT NULL,
  description text NOT NULL,
  tags        text[] DEFAULT '{}',
  created_at  timestamptz NOT NULL DEFAULT now(),
  updated_at  timestamptz NOT NULL DEFAULT now()
);

-- indexes
CREATE INDEX IF NOT EXISTS idx_chunks_doc ON chunks(document_id, idx);
CREATE INDEX IF NOT EXISTS idx_embeddings_kind_org ON embeddings(kind, org_id);
CREATE INDEX IF NOT EXISTS idx_embeddings_vector ON embeddings USING ivfflat (vector
    vector_cosine_ops) WITH (lists=100);
CREATE INDEX IF NOT EXISTS idx_matches_rfp ON matches(org_id, rfp_id, score DESC);


-- Row-level security (multitenancy)
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_rfps ON documents
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid AND kind = 'rfp');

ALTER TABLE matches ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_matches ON matches
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid);

ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_crud_offerings ON offerings
  USING (org_id = current_setting('app.org_id', true)::uuid)
```

```
86  WITH CHECK (org_id = current_setting('app.org_id', true)::uuid);
```

6.2.1 Row-Level Security (PostgreSQL)

Rationale.

Enabling Row-Level Security (RLS) with per-request/org scoping is a low-cost, high-impact guardrail for a multi-tenant MVP. The policies below anchor tenant isolation in the *database layer*, not just in API code.

- **True multi-tenancy isolation (defense in depth).** With `SET LOCAL app.org_id=...` derived from the caller's JWT, PostgreSQL itself enforces: a query cannot read or mutate rows outside its org. Even if an ORM filter or controller check is missed, the DB still blocks cross-tenant access.
- **Prevents silent data leaks.** A single missing `WHERE org_id=...` in a handler could expose other tenants' RFPs or matches. RLS makes such bugs fail *closed* instead of leaking.
- **Covers *all* paths (API & Workers).** Background workers (BullMQ) perform direct SQL. The same RLS policies apply when workers run with an explicit org scope, so batch jobs cannot cross-contaminate data.
- **Write integrity via `WITH CHECK`.** For `offerings`, the policy includes `WITH CHECK` so inserts/updates cannot spoof another org's `org_id`. The DB rejects any row where `org_id` $\neq$ current session org.
- **Principle of least privilege.** Each table exposes only the minimal actions per tenant:
  - **documents**: tenants can *read* only their own rows and only when `kind='rfp'` (prevents accidental exposure of non-RFP docs).
  - **matches**: tenants can *read* only their computed matches.
  - **offerings**: tenants can *CRUD* only their own offerings (both `USING` & `WITH CHECK` clauses).
- **Auditable and testable.** Policies are declarative SQL; integration tests run with RLS *enabled* and an org scope to catch regressions (e.g., missing policies, incorrect joins).
- **Operational safety for an MVP.** Early-stage code changes (new endpoints, ad-hoc SQL) are common and risky; RLS provides a safety net while velocity is high.
- **Works with pgvector and analytics.** Vector KNN queries still respect RLS. Any ANN/IVFFLAT/HNSW index scans are filtered by the policy, so nearest-neighbor retrieval cannot cross org boundaries.
- **Simple mental model.** App code does not need to thread `org_id` through every repository method; set the org once per request/job and let the DB enforce the contract.

Threat examples prevented.

- *Missing filter*: `SELECT * FROM matches ORDER BY score DESC LIMIT 10;` would otherwise return another tenant's data; RLS prunes rows to the current org.
- *ID guessing*: A user guessing a valid `rfp_id` cannot read it if `org_id` mismatches; the row is invisible.
- *Write spoofing*: `INSERT INTO offerings (org_id, ...)` with a foreign org is rejected by `WITH CHECK`.

Fits request/worker model.

1. API middleware/auth sets: `SET LOCAL app.org_id = $jwt.orgId;` in the transaction.
2. All subsequent `SELECT/INSERT/UPDATE` run under that scope.
3. Workers receive `orgId` in the job payload and set the same `SET LOCAL` before touching Postgres.

Net effect.

These policies make cross-tenant access *impossible by default*, reduce the blast radius of coding mistakes, and align with compliance expectations for customer data separation—while adding near-zero friction to developer workflows.

Intuitively.

Actors: *AcmeGov* (public-sector vendor team) and *BetaBank* (financial vendor team) are separate tenants using Bidion.

1. **AcmeGov uploads an RFP**. The API creates a `documents` row with `org_id=AcmeGov` and `kind='rfp'`, then workers parse → chunk → embed. Later, the `score-matches` worker writes top-N rows into `matches` with `org_id=AcmeGov`.
2. **BetaBank curates offerings**. Their team creates several `offerings` linked to `org_id=BetaBank`. These embed immediately and will *only* ever be considered inside BetaBank's tenant scope.
3. **AcmeGov views matches**. The UI calls `GET /rfps/{id}/matches`. With **RLS enabled** and `SET LOCAL app.org_id='AcmeGov'` in the request transaction, Postgres returns only rows where `org_id=AcmeGov`. BetaBank's rows are invisible at the storage layer; even a missing controller filter cannot leak them.
4. **Background jobs stay isolated**. When the `score-matches` worker re-scores an AcmeGov RFP, it sets `SET LOCAL app.org_id='AcmeGov'` before reading `documents/chunks/embeddings` and *before* writing `matches`. If a developer accidentally broadens a SQL query (e.g., drops a WHERE), RLS still prunes rows to AcmeGov only.

5. **Attack/bug averted**. Suppose a BetaBank user guesses AcmeGov's `rfp_id` and hits `/rfps/{id}/matches`. The row-level policy evaluates `org_id=current_setting('app.org_id')` and returns *no rows*. Likewise, if someone tries to `INSERT INTO offerings` with `org_id='AcmeGov'` from BetaBank's session, the `WITH CHECK` clause rejects the write.

Result: Each tenant sees only their RFPs, offerings, and computed matches. RLS makes the safe behavior the default, even during fast-moving MVP development, protecting against one-line mistakes and ID-guessing attempts.

7 Platform & Runtime

7.1 Environment & Configuration

Listing 2: .env for Supabase Postgres, Redis, Storage, OpenAI

```
# Supabase Postgres (managed)
# Use the pooled URL (port 6543) for app traffic. Keep sslmode=require.
DATABASE_URL=postgresql://postgres:<password>@db.<project-ref>.supabase.co:6543/postgres?
    sslmode=require

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379

# Storage (S3-compatible; MinIO in compose)
S3_ENDPOINT=http://localhost:9000
S3_BUCKET=bucket
S3_ACCESS_KEY=dev
S3_SECRET_KEY=devdevdev

# OpenAI
OPENAI_API_KEY=sk-...

# App
NODE_ENV=development
```

7.2 Docker Compose (Redis)

Listing 3: docker-compose.yml for Redis

```
version: "3.9"

networks:
  backend:
```

```
    driver: bridge
    internal: true   # prevents external access

volumes:
  redis_data:

services:
  redis:
    image: redis:7
    command: >
      redis-server
      --appendonly yes
      --requirepass ${REDIS_PASSWORD}
      --maxmemory-policy allkeys-lru
      --protected-mode yes
    healthcheck:
      test: ["CMD", "redis-cli", "-a", "${REDIS_PASSWORD}", "PING"]
      interval: 10s
      timeout: 3s
      retries: 5
    volumes:
      - redis_data:/data
    restart: always
    networks: [backend]
    # NOTE: do NOT publish ports in prod; keep it internal
```

7.3 Database Connection & Transactions (`lib/db/index.ts`)

Listing 4: PG Pool and transactional helper

```
import { Pool } from 'pg';
export const pool = new Pool({ connectionString: process.env.DATABASE_URL });
// Ensure DATABASE_URL includes sslmode=require for Supabase.

export async function withTx<T>(fn: (c: import('pg').PoolClient) => Promise<T>) {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    const r = await fn(client);
    await client.query('COMMIT');
    return r;
  } catch (e) {
    await client.query('ROLLBACK'); throw e;
  } finally {
```

```
    client.release();
  }
}
```

8 Shared Libraries

8.1 Text Utilities: Normalization & Chunking (`lib/shared/text.ts`)

Listing 5: Normalize strings and chunk long text

```
export function normalizeForKeywords(raw: string) {
  return raw
    .normalize('NFKC')
    .toLowerCase()
    .replace(/[^\p{L}\p{N}\s.%/-]/gu, ' ')
    .replace(/\s+/g, ' ')
    .trim();
}

// ~800-1200 tokens ≈ ~3500-5000 chars (rough)
export function chunkByChars(s: string, maxChars = 4000) {
  const parts: string[] = [];
  let buf = '';
  for (const p of s.split(/\n{2,}/)) {
    if ((buf + '\n\n' + p).length > maxChars) {
      if (buf) parts.push(buf.trim());
      if (p.length > maxChars) {
        for (let i = 0; i < p.length; i += maxChars) {
          parts.push(p.slice(i, i + maxChars));
        }
        buf = '';
      } else {
        buf = p;
      }
    } else {
      buf = buf ? `${buf}\n\n${p}` : p;
    }
  }
  if (buf) parts.push(buf.trim());
  return parts;
}
```

9 Matching Engine (v1)

9.1 Retrieval pipeline: normalization → embeddings → vector search

Purpose. Describe, once, how text from RFPs, offerings, and KB entries is converted into vectors and retrieved for scoring.

9.1.1 Pipeline overview.

1. **Text normalization & chunking**: NFKC, lowercase, preserve `.%/-`, collapse spaces; chunk by characters with small overlaps.
2. **Embeddings**: OpenAI `text-embedding-3-large` ($\mathbb{R}^{3072}$); persist to `embeddings(vector, text, org_id, kind, ref_id, chunk_id)` with pgvector index (IVFFLAT/HNSW).
3. **Vector search**: for each RFP chunk, run cosine KNN over offering (and KB) chunks using `<=>` under `vector_cosine_ops`; retrieve top-k candidates per chunk.
4. **Aggregation into scores**: pool per-chunk results into S_{sem} (top-k mean by default), combine with S_{kw} and S_{rule} in §9.2.

9.1.2 Implementation Considerations.

- **Scope.** This pipeline applies uniformly to RFP ↔ Offering, and to RFP ↔ KB retrieval used by the Assistant (RAG).
- **Consistency.** Normalization for keyword matching is identical to §9.2.3; embeddings are unaffected by normalization (they operate on raw chunk text).
- **Performance.** Use ANN indexes (IVFFLAT/HNSW) with sensible params; optionally shortlist offerings via per-chunk KNN before full scoring.

9.2 Scoring Model

We combine: *semantic*, *keyword*, and *rules*.

9.2.1 Components

- **Semantic** $S_{\text{sem}} \in [0,1]$: mean of top-k cosine similarities per offering (default $k = 3$), clamped.
- **Keyword** $S_{\text{kw}} \in [0,1]$: sum of configured boosts for normalized exact-term hits, capped at 1.
- **Rules** $S_{\text{rule}} \in [0,1]$: penalty for missing required / present forbidden, then $1 -$ penalty, clamped.

9.2.1.1 Normalization of component scores to the unit interval $[0,1]$

1. **Interpretability.** A $[0,1]$ scale reads like a probability/percentage: 0 = none, 1 = maximum.
2. **Negative cosine is not useful for semantics.** Cosine similarity lives in $[-1,1]$, but for language embeddings negative values rarely indicate "anti-meaning"; we treat them as "unrelated" $\Rightarrow 0$.

3. **Consistency with `pgvector` distance.** Under `vector_cosine_ops`, `<=>` returns cosine *distance* in $[0, 2]$. We form similarity as $1 -$ distance and then clamp to $[0, 1]$.
4. **Aggregation stability.** Averaging (e.g., top-k means) is more stable when each contribution lies in $[0, 1]$, making scores comparable across datasets.

Operationalization. We normalize each component as follows:

$$S_{\text{sem}}^{(\text{pair})} = \max\{0,\ \text{sim}_{\cos}(\mathbf{a}, \mathbf{b})\},$$

$$S_{\text{kw}} = \min\Big(1,\ \sum_t w_t\, \mathbf{1}_{\,\text{substr}(\text{normalize}(t),\, \text{normalize}(\texttt{text}))}\Big),$$

$$S_{\text{rule}} = \max\big(0,\ \min(1,\ 1 - \text{penalty})\big).$$

This enforces $S_{\text{sem}}, S_{\text{kw}}, S_{\text{rule}} \in [0, 1]$ and yields a well-behaved hybrid score.

9.2.1.2 Conceptual Interpretation of the Scoring Components

Semantic ("does it feel right?") Imagine reading two stories and asking whether they talk about the same idea. We take a few best-matching parts (top-k, usually $k = 3$), average their similarity, and keep the result between 0 and 1 ("clamped").

Keyword ("does it say the magic words?") We keep a list of important phrases (e.g., "SOC2", "24/7"). Each time the text contains one, we add some points, stopping at 1 ("capped at 1").

Rules ("did it follow must-do / must-not-do?") There are green-list terms (must have) and red-list terms (must not have). Missing green-list items and seeing red-list items both increase a penalty. The rules meter starts at 1 and goes down by that penalty, and is kept between 0 and 1.

Put simply:

- **Semantic** = "does it talk about the same idea?"
- **Keyword** = "does it say the special words?"
- **Rules** = "did it follow the simple yes/no rules?"

9.2.2 *Semantic Score* $S_{\text{sem}} \in [0, 1]$

9.2.2.1 Definition of *Semantic Score*

Semantic Score S_{sem} measures how strongly an offering discusses the same ideas as an RFP. We embed text chunks for each document using OpenAI `text-embedding-3-large` (dimension 3072), store them in Postgres `pgvector`, and compute cosine-based similarity:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\, \|\mathbf{b}\|} \in [-1, 1], \qquad d_{\cos}(\mathbf{a}, \mathbf{b}) = 1 - \text{sim}(\mathbf{a}, \mathbf{b}).$$

In pgvector (implementation): under `vector_cosine_ops`, the SQL operator `<=>` returns cosine *distance*:

$$\text{sim}(\mathbf{a}, \mathbf{b}) \ = \ 1 - \big(\mathbf{a} \texttt{ <=> } \mathbf{b}\big).$$

Aggregation uses a *top-k mean* over chunk–chunk matches (default k=3): for each RFP chunk, average its best k offering-chunk similarities, then average across RFP chunks.

- **Embeddings**: $\mathbf{v} \in \mathbb{R}^{3072}$ in `embeddings(vector)`.
- **Similarity**: cosine similarity as above; in SQL, `<=>` yields cosine distance.
- **Aggregation**: per-RFP-chunk top-k mean, then mean across RFP chunks.

9.2.2.2 Vector Search (semantic similarity)

Given an RFP *chunk* vector $\mathbf{v}$, find the most semantically similar *offering* chunks (requires an `ivfflat` or `hnsw` index):

Listing 6: Vector search for a single RFP chunk against offering embeddings (pgvector, cosine)

```
-- $1::vector is the RFP chunk vector; $2 is org_id
SELECT e.ref_id AS offering_id,
       1 - (e.vector <=> $1::vector) AS sim
FROM embeddings e
WHERE e.kind = 'offering' AND e.org_id = $2
ORDER BY e.vector <=> $1::vector     -- ascending distance = descending similarity
LIMIT 20;
```

Aggregation per RFP: loop over its chunks; for each chunk fetch top-k offering matches. Aggregate to an *offering-level* semantic score using either:

- $S_{\text{sem}}^{\max}(\text{off}) = \max_{\text{chunks}} \text{sim}$ (highlights strongest match),
- $S_{\text{sem}}^{\text{topN-mean}}(\text{off}) = \dfrac{1}{N}\sum_{\ell=1}^{N} \text{sim}_{(\ell)}$ (more robust to noise).

9.2.2.3 Semantic Score Algorithm

Let $R = \{r_1, \ldots, r_{|R|}\}$ be RFP chunks and $O = \{o_1, \ldots, o_{|O|}\}$ offering chunks, with embeddings $\mathbf{r}_i, \mathbf{o}_j \in \mathbb{R}^{3072}$.

Step 1 Compute $\text{sim}_{ij} = \text{sim}(\mathbf{r}_i, \mathbf{o}_j) \in [0, 1]$.

Step 2 Top-k: for each i, take $T_i = \text{TopK}(\{\text{sim}_{ij}\}_{j=1}^{|O|}, k)$ of size $k' = \min(k, |O|)$.

Step 3 Per-chunk mean: $m_i = \frac{1}{k'} \sum_{t \in T_i} t$.

Step 4 Across-chunks mean: $\widetilde{S}_{\text{sem}} = \frac{1}{|R|} \sum_{i=1}^{|R|} m_i$.

Step 5 Clamp: $S_{\text{sem}} = \max\big(0, \min(1, \widetilde{S}_{\text{sem}})\big)$.

Listing 7: pgvector SQL: top-k mean per RFP chunk, then average

```
-- Inputs: :rfp_id, :offering_id, :org_id, :k
WITH rfp_chunks AS (
  SELECT e.chunk_id, e.vector
```

```
  FROM embeddings e
  WHERE e.org_id = :org_id AND e.kind = 'rfp' AND e.ref_id = :rfp_id
),
off_chunks AS (
  SELECT e.chunk_id, e.vector
  FROM embeddings e
  WHERE e.org_id = :org_id AND e.kind = 'offering' AND e.ref_id = :offering_id
),
topk_per_rfp AS (
  SELECT
    r.chunk_id AS r_chunk_id,
    AVG(1 - (r.vector <=> o.vector)) AS mean_topk
  FROM rfp_chunks r
  CROSS JOIN LATERAL (
    SELECT o.vector
    FROM off_chunks o
    ORDER BY r.vector <=> o.vector
    LIMIT :k
  ) o
  GROUP BY r.chunk_id
)
SELECT GREATEST(0, LEAST(1, AVG(mean_topk))) AS s_sem
FROM topk_per_rfp;
```

9.2.2.4 Scope & Delimitations

- **Chunking**: avoid very long chunks (dilution) or very short chunks (noise). Keep stable size and small overlap.
- **Top-k**: small k is outlier-prone; large k over-smooths. Default k=3.
- **Indexing**: use `IVFFLAT`; consider `HNSW` for better recall/latency where available.
- **Explainability**: persist best snippet pairs (chunk ids + preview) into `matches.reasons`.

9.2.2.5 Example

Two RFP chunks (r_1, r_2), three offering chunks (o_1, o_2, o_3):

$$\begin{aligned}&\mathrm{sim}(r_1, o_1) = 0.88,\ \mathrm{sim}(r_1, o_2) = 0.42,\ \mathrm{sim}(r_1, o_3) = 0.66,\\ &\mathrm{sim}(r_2, o_1) = 0.35,\ \mathrm{sim}(r_2, o_2) = 0.71,\ \mathrm{sim}(r_2, o_3) = 0.62.\end{aligned}$$

With k=2: $m_1 = (0.88 + 0.66)/2 = 0.77$, $m_2 = (0.71 + 0.62)/2 = 0.665$. Then $\widetilde{S}_{\mathrm{sem}} = (0.77 + 0.665)/2 = 0.7175$, so $S_{\mathrm{sem}} = 0.7175$.

9.2.2.6 Tuning

- k: sweep $\{1, 3, 5\}$; inspect ranking stability and snippet coherence.
- **Chunk size/overlap**: try $700 \sim 1200$ chars, overlap $50 \sim 150$.

- **Index params**: tune #lists for `IVFFLAT`; for `HNSW`, tune M and ef_search.
- **Variants (optional)**: section-weighted pooling or symmetric pooling (also top-k from offering→RFP).

9.2.3 *Keyword score* $S_{\text{kw}} \in [0, 1]$

9.2.3.1 Definition of *Keyword Score*

Keyword Score S_{kw} rewards the presence of configured exact terms (after normalization) in the offering text. Each term t has a nonnegative weight w_t defined per org in `score_config.boosts`. The score is the capped sum of weights for terms found:

$$S_{\text{kw}} \;=\; \min\Big(1,\; \sum_{t \in \mathcal{T}} w_t \cdot \mathbf{1}\{\text{norm(text) contains norm}(t)\}\Big),$$

where norm$(\cdot)$ applies text normalization (NFKC, lowercase, preserve `.%/-`, collapse spaces).

Listing 8: Example per-org boosts (stored in 'score_config.boosts')

```
{
  "boosts": {
    "soc2": 0.25,
    "hipaa": 0.20,
    "24/7": 0.15,
    "99.99%": 0.10,
    "iso 27001": 0.20
  }
}
```

9.2.3.2 Keyword Score Algorithm

Let text be the offering text (title + description + relevant fields). Let Boosts $= \{(t, w_t)\}$.

Step 1 Normalize both text and each term t: NFKC → lowercase → remove non-alphanumerics except `.%/-` → collapse spaces.

Step 2 Accumulate: $s \leftarrow \sum_{(t,w_t)\in \text{Boosts}} w_t \cdot \mathbf{1}\{\text{norm(text) contains norm}(t)\}$.

Step 3 Cap to $[0, 1]$: $S_{\text{kw}} \leftarrow \min(1, s)$.

Listing 9: Keyword score with normalization and capping

```
import { normalizeForKeywords } from './text'; // NFKC -> lower -> keep .%/- -> collapse
    spaces

export function keywordScore(offTxt: string, boosts: Record<string, number>) {
  const text = normalizeForKeywords(offTxt);
  let score = 0;
  for (const [term, w] of Object.entries(boosts)) {
```

```
    if (text.includes(normalizeForKeywords(term))) score += w;
  }
  return Math.min(1, score); // S_kw in [0,1]
}
```

Listing 10: Reading boosts for an org (used by API/worker)

```
SELECT boosts
FROM score_config
WHERE org_id = $1;
```

9.2.3.3 Scope & Delimitations

- **Exact substrings after normalization**: no stemming/lemmatization by default; add aliases if needed (e.g., "soc 2", "soc ii").
- **Symbols preserved**: `-`, `%`, `/`, `.` are kept so terms like `24/7`, `99.99%`, `iso 27001` match reliably.
- **Case-insensitive**: enforced by lowercasing in normalization.
- **Explainability**: record matched terms and their weights in `matches.reasons.keyword_hits`.

9.2.3.4 Example

Config: `boosts={"soc2":0.25,"24/7":0.15,"iso 27001":0.2}`. Offering text (normalized) contains `soc2` and `24/7` but not `iso 27001`. Then

$$S_{\text{kw}} \;=\; \min\bigl(1,\; 0.25 + 0.15\bigr) \;=\; 0.40.$$

9.2.3.5 Tuning

- **Weights**: calibrate $\{w_t\}$ to reflect business-critical terms; keep $\sum w_t \leq 1$ if we want natural capping.
- **Aliases / n-grams**: add common variants ("soc 2", "soc ii") to reduce phrasing brittleness; optionally introduce light lemmatization.
- **Noise control**: prefer a concise list of high-signal terms; revisit weights using offline evals and error analysis.

9.2.4 *Rules* $S_{\text{rule}} \in [0, 1]$

9.2.4.1 Definition of *Rules*

Rules are non-semantic constraints that adjust the score using simple, configurable checks stored per org in `score_config`:

- **Required terms** (`required[]`): phrases that *must* appear in the offering text. Missing terms increase a penalty.
- **Forbidden terms** (`forbidden[]`): phrases that *must not* appear. Presence increases a penalty.

Listing 11: Rule-related columns in `score_config`

```
-- Per-org configuration of rules and keyword boosts
-- (Boosts are used by the keyword component, not the rules component.)
org_id    uuid  PRIMARY KEY,
boosts    jsonb NOT NULL DEFAULT '{}'::jsonb,  -- e.g., {"soc2":0.25,"24/7":0.15}
required  jsonb NOT NULL DEFAULT '[]'::jsonb,  -- e.g., ["soc2","99.99% uptime"]
forbidden jsonb NOT NULL DEFAULT '[]'::jsonb   -- e.g., ["on-prem only"]
```

9.2.4.2 Rule Score Algorithm

Let the (normalized) offering text be compared against configured terms:

- **Normalize** both text and terms: NFKC, lowercase, strip non-alphanumerics except `.%/-`, collapse spaces.
- **Measure misses/hits**:
  - `miss` = count of required terms not found
  - `forb` = count of forbidden terms found
- **Convert to proportions**:
  - $p_{\text{Req}} = \texttt{miss} \,/\, |\text{required}|$ (or 0 if no required terms)
  - $p_{\text{For}} = \texttt{forb} \,/\, |\text{forbidden}|$ (or 0 if no forbidden terms)
- **Weighted penalty**: $\text{penalty} = 0.7\, p_{\text{Req}} + 0.3\, p_{\text{For}}$.
- **Rule score**: $S_{\text{rule}} = \max(0,\, \min(1,\, 1 - \text{penalty}))$.

Missing required terms is penalized more than finding forbidden terms by design (0.7 vs. 0.3).

Listing 12: JavaScript design for rule score computation

```
import { normalizeForKeywords } from './text';

function ruleCompliance(text: string, required: string[], forbidden: string[]) {
  const norm = normalizeForKeywords(text);
  const miss = required.filter(r => !norm.includes(normalizeForKeywords(r))).length;
  const forb = forbidden.filter(f => norm.includes(normalizeForKeywords(f))).length;
  const pReq = required.length ? miss / required.length : 0;
  const pFor = forbidden.length ? forb / forbidden.length : 0;
  const penalty = 0.7 * pReq + 0.3 * pFor;
  return Math.max(0, Math.min(1, 1 - penalty));
}
```

9.2.4.3 Scope & Delimitations

- **Exact substrings after normalization**: no stemming/lemmatization. Consider synonyms if needed.
- **Symbols preserved**: hyphens, %, and slashes survive normalization, so `24/7`, `99.99%` still match.
- **Case-insensitive** by design (lowercased).
- **Explainability**: we persist `required_missing` and `forbidden_hit` in `matches.reasons`.

9.2.4.4 Example

Config: `required=["soc2","99.99% uptime"]`, `forbidden=["on-prem only"]`.
Offering contains `SOC2`, lacks `99.99% uptime`, and does not contain `on-prem only`. Then:

$$p_{\text{Req}} = \frac{1}{2} = 0.5, \quad p_{\text{For}} = 0, \quad \text{penalty} = 0.7 \cdot 0.5 + 0.3 \cdot 0 = 0.35, \quad S_{\text{rule}} = 0.65.$$

9.2.4.5 Tuning

- **Hard gates**: increase the final weight of rules (e.g., 0.6/0.2/0.2) or pre-filter offerings on critical must-haves.
- **Reduce false negatives**: add synonyms in `required[]` or switch to a light fuzzy/regex matcher.

9.2.5 Final Score

$$S_{\text{final}} \;=\; 0.7\, S_{\text{sem}} \;+\; 0.2\, S_{\text{kw}} \;+\; 0.1\, S_{\text{rule}}$$

clamped to $[0, 1]$.

9.2.6 Example Calculation of $S_{\text{sem}}, S_{\text{kw}}, S_{\text{rule}}$ and S_{final}

9.2.6.1 RFP extract (single chunk).

"Looking for **SOC2 Type II** compliant **cloud hosting** with **24/7 support** and **99.99% uptime**."

9.2.6.2 Offering A (title + description).

"Cloud Hosting (Enterprise). **SOC2 Type II**, **24/7 support**, SLA **99.99%**."
Component scores:

- S_{sem}: from pgvector (e.g., max across RFP chunks) $\rightarrow$ **0.84**.
- S_{kw}: term hits: `soc2`(0.25) + `24/7`(0.15) + `99.99%`(0.10) $\rightarrow$ **0.50** (capped ≤ 1).
- S_{rule}: `required`={"soc2", "24/7"}, `forbidden`={"on-prem only"}.
  miss=0, hitForbidden=0 $\Rightarrow$ penalty $= 0 \Rightarrow \mathbf{S_{\text{rule}} = 1.00}$.

Final score:

$$S_{\text{final}} = 0.7 \cdot 0.84 \;+\; 0.2 \cdot 0.50 \;+\; 0.1 \cdot 1.00 = 0.588 + 0.10 + 0.10 = \mathbf{0.788}.$$

9.2.6.3 Offering B (title + description).

"Budget VPS. No compliance guarantees."

Component scores:

- $S_{\text{sem}} \approx \mathbf{0.22}$.
- $S_{\text{kw}} = \mathbf{0.00}$.
- S_{rule}: same `required`; miss= $2/2 \Rightarrow p_{\text{Req}} = 1$, penalty= $0.7 \Rightarrow \mathbf{S}_{\text{rule}} = \mathbf{0.30}$.

Final score:

$$S_{\text{final}} = 0.7 \cdot 0.22 \ + \ 0.2 \cdot 0 \ + \ 0.1 \cdot 0.30 = 0.154 + 0 + 0.03 = \mathbf{0.184}.$$

9.2.6.4 Ranking.

Offering A (0.788) $\gg$ Offering B (0.184).

9.3 Worker Task: Score Matches (`apps/worker/src/scoreMatches.task.ts`)

The `scoreMatches` worker is the component that *materializes* a match result from previously embedded data: it computes the hybrid score (semantic similarity + keyword boosts + rule compliance) and persists ranked `matches` for each RFP. In other words, it is the core execution unit of the matching engine's scoring stage: taking vectorized RFP/Offering representations and turning them into actionable, ranked results that drive the UI and downstream workflow. Background queues and workers are used only as an execution model; the algorithmic logic (*how* we score) is what binds this task to the Matching Engine (v1) rather than to generic job infrastructure.

9.3.1 Design process of the worker

1. **Inputs.** Receives `{ rfpId, orgId }` from the queue once embeddings for the RFP are ready.
2. **Candidate set.** Pulls all offerings for the organization (`offerings.id`) as matching candidates.
3. **Semantic score.** For each candidate, computes cosine similarity between the offering's vectors and the RFP's vectors using `pgvector`. To increase robustness and reduce outlier effects, aggregates as the mean of top-k similarities (default k=3).
4. **Lexical features.** Reads `score_config` (boosts, required, forbidden) and computes (a) keyword score via normalized exact-term hits and (b) rule compliance as 1−penalty for missing required/present forbidden terms.
5. **Hybrid scoring.** Combines features via a simple, transparent formula $S = 0.7\,S_{\text{sem}} + 0.2\,S_{\text{kw}} + 0.1\,S_{\text{rule}}$, clamped to $[0, 1]$.
6. **Explanations.** Captures the top semantic snippet and lists keyword/rule hits into `reasons` (JSON) for "why it matched."
7. **Persistence.** Deletes prior rows for `(org_id, rfp_id)` and inserts the top-N (default 10) matches into `matches`, wrapped in a DB transaction for atomicity.

8. **Idempotency & runtime.** The worker is safe to re-run for the same `rfpId` (previous results are replaced), and concurrency is controlled at the queue level.

Listing 13: Cosine KNN + keyword/rules to rank offerings

```
import { pool } from '../../../lib/db';
import { keywordScore, ruleCompliance, finalScore } from '../../../lib/shared/scoring';

async function getScoreConfig(orgId: string) {
  const { rows } = await pool.query(
    `SELECT boosts, required, forbidden FROM score_config WHERE org_id=$1`, [orgId]
  );
  if (!rows[0]) return { boosts:{}, required:[], forbidden:[] };
  const { boosts, required, forbidden } = rows[0];
  return { boosts, required, forbidden };
}

async function semanticForOffering(orgId: string, rfpId: string, offeringId: string, k =
    3): Promise<number> {
  const { rows } = await pool.query(`
    WITH rfp_chunks AS (
      SELECT vector
      FROM embeddings
      WHERE org_id = $1 AND kind = 'rfp' AND ref_id = $2
    ),
    off_chunks AS (
      SELECT vector
      FROM embeddings
      WHERE org_id = $1 AND kind = 'offering' AND ref_id = $3
    ),
    topk_per_rfp AS (
      SELECT AVG(1 - (r.vector <=> o.vector)) AS mean_topk
      FROM rfp_chunks r
      CROSS JOIN LATERAL (
        SELECT o.vector
        FROM off_chunks o
        ORDER BY r.vector <=> o.vector
        LIMIT $4
      ) o
      GROUP BY r.vector
    )
    SELECT GREATEST(0, LEAST(1, AVG(mean_topk))) AS s_sem
    FROM topk_per_rfp
  `, [orgId, rfpId, offeringId, k]);

```

```
  return Number(rows[0]?.s_sem ?? 0);
}

async function offeringText(offeringId: string) {
  const { rows } = await pool.query(
    `SELECT title, description FROM offerings WHERE id=$1`, [offeringId]
  );
  if (!rows[0]) return '';
  return `${rows[0].title}\n\n${rows[0].description}`;
}

export async function scoreMatchesTask({ rfpId, orgId }:{ rfpId:string; orgId:string }) {
  const offs = await pool.query(`SELECT id FROM offerings WHERE org_id=$1`, [orgId]);
  const cfg = await getScoreConfig(orgId);

  const results: Array<{ offering_id:string; score:number; reasons:any }> = [];

  for (const o of offs.rows) {
    const offId = o.id as string;
    const sem  = await semanticForOffering(orgId, rfpId, offId);
    const text = await offeringText(offId);
    const kw   = keywordScore(text, cfg.boosts);
    const rule = ruleCompliance(text, cfg.required, cfg.forbidden);
    const score= finalScore(sem, kw, rule);

    const topChunk = await pool.query(`
      SELECT e.text, (1 - (e.vector <=> r.vector)) AS sim
      FROM embeddings e
      JOIN embeddings r ON r.ref_id=$1 AND r.kind='rfp' AND r.org_id=$2
      WHERE e.ref_id=$3 AND e.kind='offering' AND e.org_id=$2
      ORDER BY sim DESC
      LIMIT 1
    `, [rfpId, orgId, offId]);

    results.push({
      offering_id: offId,
      score,
      reasons: {
        semantic_top_sim: Number(topChunk.rows[0]?.sim ?? 0),
        semantic_snippet: topChunk.rows[0]?.text ?? null,
        keyword_hits: Object.keys(cfg.boosts).filter(k => text.toLowerCase().includes(k.
    toLowerCase())),
        required_missing: cfg.required.filter(r => !text.toLowerCase().includes(r.
    toLowerCase())),
```

```
        forbidden_hit: cfg.forbidden.filter(f => text.toLowerCase().includes(f.toLowerCase())),
      }
    });
  }

  results.sort((a,b)=>b.score - a.score);
  const top = results.slice(0,10);
  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    await client.query(`DELETE FROM matches WHERE org_id=$1 AND rfp_id=$2`, [orgId, rfpId]);
    for (const r of top) {
      await client.query(`
        INSERT INTO matches(org_id, rfp_id, offering_id, score, reasons)
        VALUES ($1,$2,$3,$4,$5)
      `, [orgId, rfpId, r.offering_id, r.score, r.reasons]);
    }
    await client.query('COMMIT');
  } catch (e) { await client.query('ROLLBACK'); throw e; }
  finally { client.release(); }

  return { count: top.length };
}
```

10 Background Jobs Queue & Worker Chain Architecture Design

Queues (•): `parse-doc`, `embed-chunks`, `score-matches`, optional `sync-source`.

Why a Queue & Worker Chain (and not full EDA) — Rationale.

Our Bidion MVP pipeline is *causally ordered*: **Parse** must complete before **Embed**, and **Embed** must complete before **Score**. A *job queue + workers* model gives us:

1. **Deterministic sequencing & backpressure.** The queue enforces order-of-operations and naturally buffers spikes so upstream API traffic does not overload downstream CPU/LLM work.
2. **Isolated retries & idempotency.** Each step can retry independently with idempotent writes (UPSERTs / delete+insert in a transaction), without replaying the entire pipeline.
3. **Simple scaling knobs.** WE scale per-worker concurrency and replica counts by queue (e.g., higher concurrency for embeddings than parsing).
4. **Operational clarity.** Success/failure is tracked at the job boundary (*attempts, backoff, DLQ*), which maps cleanly to on-call runbooks and SLOs.

By contrast, a *full event-driven architecture (EDA)* shines when a single business *fact* should fan out to many *independent* reactions in parallel (e.g., "ImageUploaded" → resize, caption, audit, notify). Our core pipeline is *not* that pattern; it is a *dependent chain*. Using EDA for the core would add coordination complexity (ordering, deduplication, replay windows) without commensurate benefit.

Where EDA still fits.

Even with a job chain, WE can publish lightweight "facts" at the boundaries to enable *independent* side effects without coupling them to the core:

- After Parse commits: emit `DocumentParsed` (analytics, audit, notifications can subscribe).
- After Embeddings persist: emit `EmbeddingsReady` (metrics, cache warmers, index sync).
- After Scoring commits: emit `MatchesScored` (webhooks, dashboards, alerts).

Implementation options: Redis Pub/Sub (best-effort broadcast) or Redis Streams (+ consumer groups) with an *outbox* table to ensure "commit then publish" integrity. This preserves the deterministic chain for the *must-be-ordered* steps while unlocking event fan-out for *nice-to-have* side effects.

Evolution path (future-friendly).

If future product needs require broad fan-out, cross-service decoupling, or long retention/replay, WE can:

1. Keep the ordered chain for *parse* → *embed* → *score*.
2. Graduate side-effect events from Redis to a durable log (e.g., Kafka/Pulsar).
3. Promote consumers (audit, webhooks, analytics) to independent services subscribing to those facts.

10.1 Worker Sequence

1. **ParseDocWorker**: fetch file → extract text → chunk → insert `chunks`.
2. **EmbedChunksWorker**: batch embed → insert `embeddings` (RLS scope set via `app.org_id`).
3. **ScoreMatchesWorker**: for RFP, compute hybrid scores vs offerings → write `matches` (RLS scope set via `app.org_id`).

10.2 Job Infrastructure: Queues & Types (`apps/worker/src/queue.ts`)

Listing 14: BullMQ queue setup and job payloads (RLS-ready)

```
import { Queue, QueueEvents, JobsOptions } from 'bullmq';
import IORedis from 'ioredis';

export const redis = new IORedis(process.env.REDIS_URL ?? {
  host: process.env.REDIS_HOST,
  port: Number(process.env.REDIS_PORT || 6379),
```

```
  password: process.env.REDIS_PASSWORD,
  tls: process.env.REDIS_TLS === '1' ? {} : undefined,
} as any);

export function makeQueue(name: string) {
  const q = new Queue(name, { connection: redis });
  const events = new QueueEvents(name, { connection: redis });
  return { q, events };
}

// Include orgId in all payloads that will touch RLS-protected tables
export type ParseDocPayload     = { documentId: string; orgId: string };
export type EmbedChunksPayload  = { documentId: string; orgId: string; kind: 'rfp'|'kb'|'
    offering' };
export type ScoreMatchesPayload = { rfpId: string; orgId: string };

export const queues = {
  parseDoc:     makeQueue('parse-doc'),
  embedChunks:  makeQueue('embed-chunks'),
  scoreMatches: makeQueue('score-matches'),
};

// Align with spec: attempts=5, exponential backoff base 250ms
export const defaultJobOpts: JobsOptions = {
  attempts: 5,
  backoff: { type: 'exponential', delay: 250 },
  removeOnComplete: { age: 86400 },
  removeOnFail: { age: 86400 },
};
```

10.3 OpenAI Embeddings Adapter (`apps/worker/src/embeddings.ts`)

Listing 15: Batch embed texts with OpenAI

```
import OpenAI from 'openai';
const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });

export async function embedBatch(texts: string[]) {
  const res = await client.embeddings.create({
    model: 'text-embedding-3-large',
    input: texts
  });
  return res.data.map(d => d.embedding);
}
```

10.4 Worker Task: Parse Documents (`apps/worker/src/parseDoc.task.ts`)

Listing 16: Extract text, chunk, enqueue embeddings (RLS-safe)

```
import { pool } from '../../../lib/db';
import { chunkByChars } from '../../../lib/shared/text';
import { queues, defaultJobOpts } from './queue';
import pdf from 'pdf-parse';

// Node 18+ has global fetch; avoid extra deps
async function fetchFile(storageUrl: string): Promise<Buffer> {
  const r = await fetch(storageUrl);
  if (!r.ok) throw new Error(`fetch failed: ${r.status}`);
  return Buffer.from(await r.arrayBuffer());
}

export async function parseDoc({ documentId, orgId }: { documentId: string; orgId: string
    }) {
  // Use a single client/tx so we can SET LOCAL app.org_id per the RLS policy
  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    await client.query('SET LOCAL app.org_id = $1', [orgId]);

    const { rows } = await client.query(
      `SELECT id, org_id, kind, storage_url, mime FROM documents WHERE id=$1`,
      [documentId]
    );
    if (!rows[0]) throw new Error('doc not found');
    const { storage_url, mime, kind } = rows[0];

    const buf = await fetchFile(storage_url);
    let text = '';
    if ((mime || '').includes('pdf')) {
      const data = await pdf(buf);
      text = data.text || '';
    } else {
      text = buf.toString('utf8');
    }

    const parts = chunkByChars(text, 4000);
    for (let i = 0; i < parts.length; i++) {
      await client.query(
        `INSERT INTO chunks(document_id, idx, text) VALUES ($1, $2, $3)`,
        [documentId, i, parts[i]]
```

```
      );
    }

    await client.query('COMMIT');

    // Enqueue embedding with org/kind for downstream RLS + branching
    await queues.embedChunks.q.add('embed', { documentId, orgId, kind }, defaultJobOpts);
    return { chunks: parts.length };
  } catch (e) {
    await client.query('ROLLBACK');
    throw e;
  } finally {
    client.release();
  }
}
```

10.5 Worker Task: Embed Chunks (`apps/worker/src/embedChunks.task.ts`)

Listing 17: Embed chunk texts and persist vectors (RLS-safe)

```
import { pool } from '../../../lib/db';
import { embedBatch } from './embeddings';
import { queues, defaultJobOpts } from './queue';

export async function embedChunksTask(
  { documentId, orgId, kind }: { documentId: string; orgId: string; kind: 'rfp'|'kb'|'
    offering' }
) {
  // chunks table is not RLS-protected; embeddings is. We'll set RLS for writes.
  const { rows: chunkRows } = await pool.query(
    `SELECT id, text FROM chunks WHERE document_id=$1 ORDER BY idx`,
    [documentId]
  );

  const batchSize = 64;
  for (let i = 0; i < chunkRows.length; i += batchSize) {
    const slice = chunkRows.slice(i, i + batchSize);
    const vecs = await embedBatch(slice.map(r => r.text));
    const textAndVec = slice.map((r, j) => ({ chunkId: r.id, text: r.text, vec: vecs[j]
    }));

    const client = await pool.connect();
    try {
      await client.query('BEGIN');
```

```
      await client.query('SET LOCAL app.org_id = $1', [orgId]); // RLS scope for embeddings

      for (const tv of textAndVec) {
        await client.query(
          `INSERT INTO embeddings(org_id, kind, ref_id, chunk_id, vector, text)
           VALUES ($1,$2,$3,$4,$5::vector,$6)
           ON CONFLICT (org_id, kind, ref_id, chunk_id)
           DO UPDATE SET vector=EXCLUDED.vector, text=EXCLUDED.text`,
          [orgId, kind, documentId, tv.chunkId, `[${tv.vec.join(',')}]`, tv.text]
        );
      }
      await client.query('COMMIT');
    } catch (e) {
      await client.query('ROLLBACK'); throw e;
    } finally {
      client.release();
    }
  }

  if (kind === 'rfp') {
    await queues.scoreMatches.q.add('score', { rfpId: documentId, orgId }, defaultJobOpts);
  }
}
```

10.6 Worker Entrypoint: Concurrency & Startup (`apps/worker/src/main.ts`)

Listing 18: Start BullMQ workers with concurrency

```
import { Worker } from 'bullmq';
import { redis } from './queue';
import { parseDoc } from './parseDoc.task';
import { embedChunksTask } from './embedChunks.task';
import { scoreMatchesTask } from './scoreMatches.task';

new Worker('parse-doc', async job => parseDoc(job.data), { connection: redis, concurrency: 4 });
new Worker('embed-chunks', async job => embedChunksTask(job.data), { connection: redis, concurrency: 8 });
new Worker('score-matches', async job => scoreMatchesTask(job.data), { connection: redis, concurrency: 4 });

console.log('Workers up: parse-doc, embed-chunks, score-matches');
```

10.7 Operational Resilience

10.7.1 Delivery semantics & idempotency

10.7.1.1 Reference enqueue + idempotent handler

Listing 19: apps/worker/src/score.enqueue.ts

```ts
import type { JobsOptions } from 'bullmq';
import { queues } from './queue';
import { Client as Pg } from 'pg';

/** In-queue de-dup: jobId = orgId:rfpId */
export async function enqueueReScore(orgId: string, rfpId: string) {
  const opts: JobsOptions = {
    jobId: `${orgId}:${rfpId}`,
    attempts: 5,
    backoff: { type: 'exponential', delay: 250 },
    removeOnComplete: { age: 86400 },
    removeOnFail: { age: 86400 },
  };
  return queues.scoreMatches.q.add('re-score', { orgId, rfpId }, opts);
}

/** Idempotent writer: delete+insert or UPSERT inside a tx */
export async function recomputeMatches(pg: Pg, orgId: string, rfpId: string) {
  await pg.query('BEGIN');
  try {
    await pg.query('SET LOCAL app.org_id = $1', [orgId]); // RLS scope
    await pg.query('DELETE FROM matches WHERE org_id=$1 AND rfp_id=$2', [orgId, rfpId]);

    // ... compute rows
    const rows = [{ offering_id: 'off-1', score: 0.87 }];

    const sql = `
      INSERT INTO matches (org_id, rfp_id, offering_id, score, updated_at)
      VALUES ($1,$2,$3,$4,now())
      ON CONFLICT (org_id,rfp_id,offering_id)
      DO UPDATE SET score=EXCLUDED.score, updated_at=now()
    `;
    for (const r of rows) {
      await pg.query(sql, [orgId, rrfpId, r.offering_id, r.score]); // <-- NOTE: ensure
    unique constraint exists
    }
    await pg.query('COMMIT');
  } catch (e) {
```

```
    await pg.query('ROLLBACK');
    throw e;
  }
}
```

10.7.1.2 Worker wiring (graceful drain compatible)

Listing 20: apps/worker/src/score.worker.ts

```
import { Worker, QueueEvents } from 'bullmq';
import { redis, queues } from './queue';
import { Client as Pg } from 'pg';
import { recomputeMatches } from './score.enqueue';

const events = new QueueEvents('score-matches', { connection: redis });

const pg = new Pg({ connectionString: process.env.DATABASE_URL! });
await pg.connect();

export const scoreWorker = new Worker('score-matches', async (job) => {
  const { orgId, rfpId } = job.data as { orgId: string; rfpId: string };
  await recomputeMatches(pg, orgId, rfpId);
  // tiny jitter to reduce herds
  await new Promise(r => setTimeout(r, Math.random() * 50));
  return { ok: true };
}, { connection: redis, concurrency: 4 });

process.on('SIGTERM', async () => {
  await queues.scoreMatches.q.pause(true); // global pause
  await scoreWorker.close();               // finish in-flight
  await events.close();
  await pg.end();
});
```

10.7.2 Retries/backoff & budgets

10.7.2.1 Retry/backoff (Worker options) + DLQ forwarding

Listing 21: apps/worker/src/retry-dlq.ts

```
import { Worker, Queue } from 'bullmq';
import { redis } from './queue';

export const dlq = new Queue('dlq:score-matches', { connection: redis });

export function withRetry(workerName: string, handler: (job:any)=>Promise<any>) {
```

```
  const w = new Worker(workerName, handler, {
    connection: redis,
    concurrency: 8,
    // attempts/backoff are typically set per-job; this is a safe default if missing
    settings: { backoffStrategies: {} }
  });

  // Route exhausted failures to DLQ
  w.on('failed', async (job, err) => {
    if (!job) return;
    const max = job.opts.attempts ?? 1;
    if (job.attemptsMade >= max) {
      await dlq.add(`dead-${workerName}`, { data: job.data, reason: err?.message }, {
        removeOnComplete: { age: 86400 }, removeOnFail: { age: 86400 }
      });
    }
  });
  return w;
}
```

10.7.2.2 Per-org inflight and LLM-QPS budgets (Redis gates)

Listing 22: apps/worker/src/budgets.ts

```
import { redis } from './queue';

// Inflight budget: allow up to limit outstanding for org
export async function beginInflight(orgId: string, limit = 500) {
  const key = `inflight:${orgId}`;
  const n = await redis.incr(key);
  if (n === 1) await redis.expire(key, 60);  // safety TTL
  if (n > limit) { await redis.decr(key); throw new Error('inflight-budget-exceeded'); }
  return () => redis.decr(key); // endInflight
}

// QPS budget (naive token bucket per org over 1s)
export async function checkLlmQps(orgId: string, orgLimit = 5, globalLimit = 100) {
  const now = Math.floor(Date.now() / 1000);
  const orgKey = `qps:${orgId}:${now}`;
  const gKey   = `qps:global:${now}`;

  // Keep it simple/accurate: two separate INCRs with TTL when counter is new
  const orgCount = await redis.incr(orgKey);
  if (orgCount === 1) await redis.expire(orgKey, 2);

  const gCount = await redis.incr(gKey);
```

```
  if (gCount === 1) await redis.expire(gKey, 2);

  if (orgCount > orgLimit)  throw new Error('llm-qps-org-exceeded');
  if (gCount > globalLimit) throw new Error('llm-qps-global-exceeded');
}
```

10.7.2.3 Using budgets in a worker

Listing 23: apps/worker/src/embed.worker.ts

```
import { Worker } from 'bullmq';
import { redis } from './queue';
import { beginInflight, checkLlmQps } from './budgets';

export const embedWorker = new Worker('embed-chunks', async (job) => {
  const { orgId } = job.data as { orgId: string };
  const end = await beginInflight(orgId, 500);
  try {
    await checkLlmQps(orgId, 5, 100);
    // ... call embeddings provider here ...
    return { ok: true };
  } finally {
    await end();
  }
}, { connection: redis, concurrency: 8 });
```

10.7.3 Kill switches

10.7.3.1 Global pause/resume endpoints (API)

Listing 24: apps/api/src/controllers/debug.queue.ts

```
import { Controller, Post, Param, NotFoundException } from '@nestjs/common';
import { queues } from '@bidion/worker/queue';

const MAP: Record<string, keyof typeof queues> = {
  'score-matches': 'scoreMatches',
  'embed-chunks': 'embedChunks',
  'parse-doc': 'parseDoc',
};

@Controller('debug/queues')
export class DebugQueueController {
  @Post(':name/pause')
  async pause(@Param('name') name: string) {
    const key = MAP[name];
```

```
    if (!key) throw new NotFoundException('unknown queue');
    await queues[key].q.pause(true);
    return { paused: true, queue: name };
  }

  @Post(':name/resume')
  async resume(@Param('name') name: string) {
    const key = MAP[name];
    if (!key) throw new NotFoundException('unknown queue');
    await queues[key].q.resume();
    return { resumed: true, queue: name };
  }
}
```

10.7.3.2 Per-tenant pause (worker-side guard)

Listing 25: apps/worker/src/guards.ts

```
import { redis } from './queue';

export async function ensureTenantNotPaused(orgId: string) {
  if (await redis.get(`ks:org:${orgId}`)) {
    throw new Error('tenant-paused');
  }
}
```

10.7.3.3 Simple circuit breaker for LLM calls (60s open)

Listing 26: apps/worker/src/circuit.ts

```
type State = 'closed'|'open'|'half';
let state: State = 'closed';
let openedAt = 0;

export async function withBreaker<T>(fn: () => Promise<T>): Promise<T> {
  const now = Date.now();
  if (state === 'open') {
    if (now - openedAt < 60_000) throw new Error('breaker-open');
    state = 'half';
  }
  try {
    const v = await fn();
    state = 'closed';
    return v;
  } catch (e: any) {
    // open on 429/5xx-like errors
```

```
    if (e?.status && (e.status === 429 || e.status >= 500)) {
      state = 'open'; openedAt = now;
    }
    throw e;
  }
}
```

10.7.4 Worker SLOs & autoscale triggers

10.7.4.1 Emit SLO metrics (Prometheus) and backlog gauges

Listing 27: apps/worker/src/metrics.ts

```
import client from 'prom-client';
import { Queue } from 'bullmq';
import { redis } from './queue';

export const registry = new client.Registry();
export const jobLatency = new client.Histogram({
  name: 'bidion_job_latency_seconds',
  help: 'Worker job latency',
  buckets: [0.5, 1, 2, 5, 10, 30, 60],
  labelNames: ['queue','name']
});
export const backlogGauge = new client.Gauge({
  name: 'bidion_queue_backlog',
  help: 'Queue backlog (waiting jobs)',
  labelNames: ['queue']
});
registry.registerMetric(jobLatency);
registry.registerMetric(backlogGauge);

export async function scrapeBacklog(queueName: string) {
  const q = new Queue(queueName, { connection: redis });
  const counts = await q.getJobCounts('waiting');
  backlogGauge.set({ queue: queueName }, counts.waiting || 0);
}

export function recordLatency(queue: string, name: string, ms: number) {
  jobLatency.observe({ queue, name }, ms / 1000);
}
```

10.7.4.2 Autoscale decision (same thresholds as tests)

Listing 28: apps/worker/src/autoscale.ts

```ts
export function decideScale(
  s: { backlog: number; minutesOver: number; avgWaitSec: number }
): { action: 'scale-up'|'hold'|'scale-down'; reason: string } {
  if ((s.backlog > 1000 && s.minutesOver >= 5) || s.avgWaitSec > 30) {
    return { action: 'scale-up', reason: 'backlog/latency over threshold' };
  }
  if (s.backlog < 200 && s.avgWaitSec < 5) {
    return { action: 'scale-down', reason: 'idle' };
  }
  return { action: 'hold', reason: 'within band' };
}
```

10.7.4.3 KEDA ScaledObject (illustrative)

Listing 29: keda-scaledobject.yaml

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: worker-embed-autoscale
spec:
  scaleTargetRef:
    name: worker-embed-deployment
  cooldownPeriod: 60
  minReplicaCount: 2
  maxReplicaCount: 20
  triggers:
    - type: redis
      metadata:
        # Prefer a secret for the address; include auth/tls as needed
        address: REDIS_HOST:6379
        # BullMQ (lists mode) waiting list key:
        # note: BullMQ uses "bull:<queue>:wait" for lists mode.
        listName: bull:embed-chunks:wait
        listLength: "1000"
# For Redis Streams mode, use the redis-streams scaler and the stream key instead.
```

11 API Design

11.1 API Server Bootstrap (`apps/api/src/server.ts`)

Listing 30: Minimal Express setup with JSON body parsing

```ts
import express from 'express';
```

```
import bodyParser from 'body-parser';
import { pool } from '../../lib/db';
import { queues, defaultJobOpts } from '../worker-bridge';
import { draftAssistant } from './svc.assistant';

const app = express();
app.use(bodyParser.json({ limit: '10mb' }));

// ... API endpoints here ...

app.listen(3001, () => console.log('API listening on :3001'));
```

11.2 RFP Endpoints

11.2.1 POST /rfps — Create Manual-Text RFP

Listing 31: Express route: POST /rfps (manual text RFP)

```
app.post('/rfps', async (req, res) => {
  // Derive orgId from authenticated user/session; do NOT trust client input
  const orgId = req.user.orgId;
  const { title, bodyText } = req.body;

  const { rows } = await pool.query(
    `INSERT INTO documents(org_id, kind, filename, storage_url, mime)
     VALUES ($1,'rfp',$2,$3,$4) RETURNING id`,
    [
      orgId,
      title ?? 'rfp.txt',
      `data:text/plain;base64,${Buffer.from(bodyText || '').toString('base64')}`,
      'text/plain'
    ]
  );

  const documentId = rows[0].id;

  // Seed initial chunk (idx 0) for manual-text path
  await pool.query(
    `INSERT INTO chunks(document_id, idx, text) VALUES ($1,0,$2)`,
    [documentId, bodyText || '']
  );

  // Enqueue embeddings with orgId and kind for downstream RLS + branching
  await queues.embedChunks.add('embed', { documentId, orgId, kind: 'rfp' },
    defaultJobOpts);
```

```
27
28    res.json({ documentId });
29  });
```

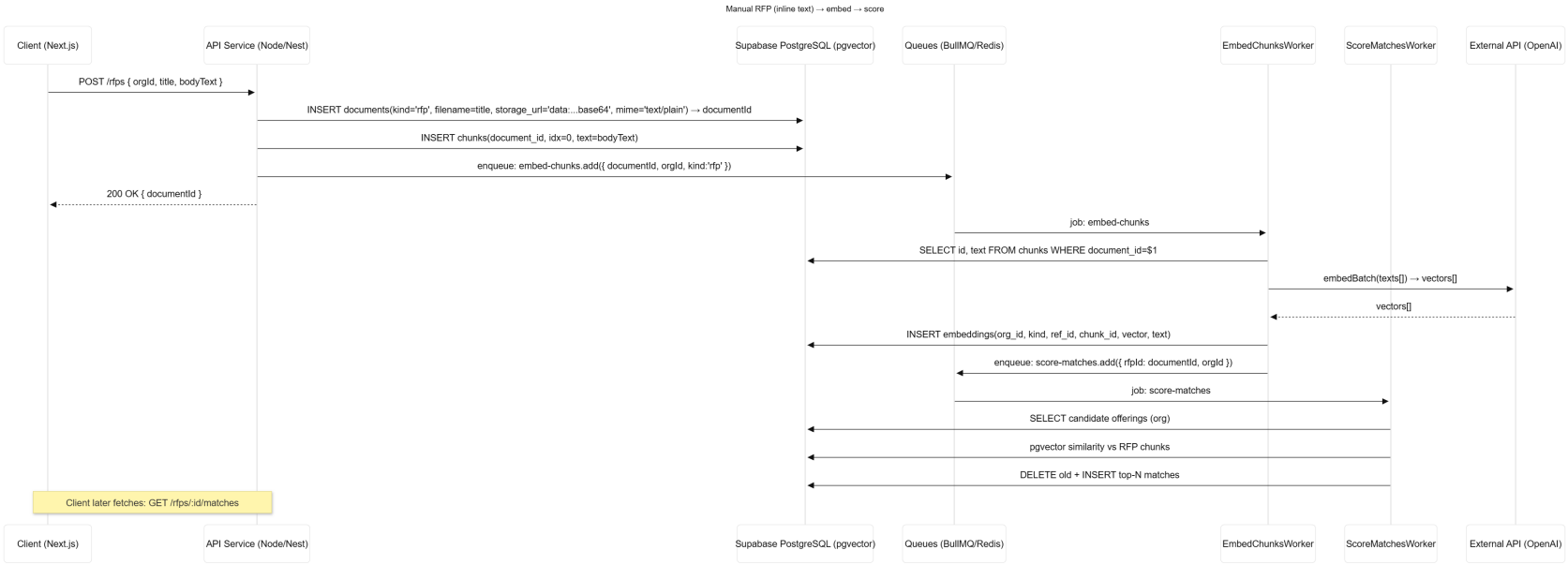


Figure 1: RFP File Upload → Parse → Embed → Score (worker-chained)

11.2.2 POST `/rfps/upload` — Register File-Backed RFP

Listing 32: Express route: POST /rfps/upload (file-backed RFP)

```
app.post('/rfps/upload', async (req,res) => {
  // Derive orgId from authenticated user/session; do NOT trust client input
  const orgId = req.user.orgId;
  const { filename, storageUrl, mime } = req.body; // storageUrl is objectPath from
    signed upload

  const { rows } = await pool.query(
    `INSERT INTO documents(org_id, kind, filename, storage_url, mime)
     VALUES ($1,'rfp',$2,$3,$4) RETURNING id`,
    [orgId, filename, storageUrl, mime]
  );

  const documentId = rows[0].id;

  // Start with parse step for file-backed path; pass orgId for RLS scoping in worker
  await queues.parseDoc.add('parse', { documentId, orgId }, defaultJobOpts);

  res.json({ documentId });
});
```

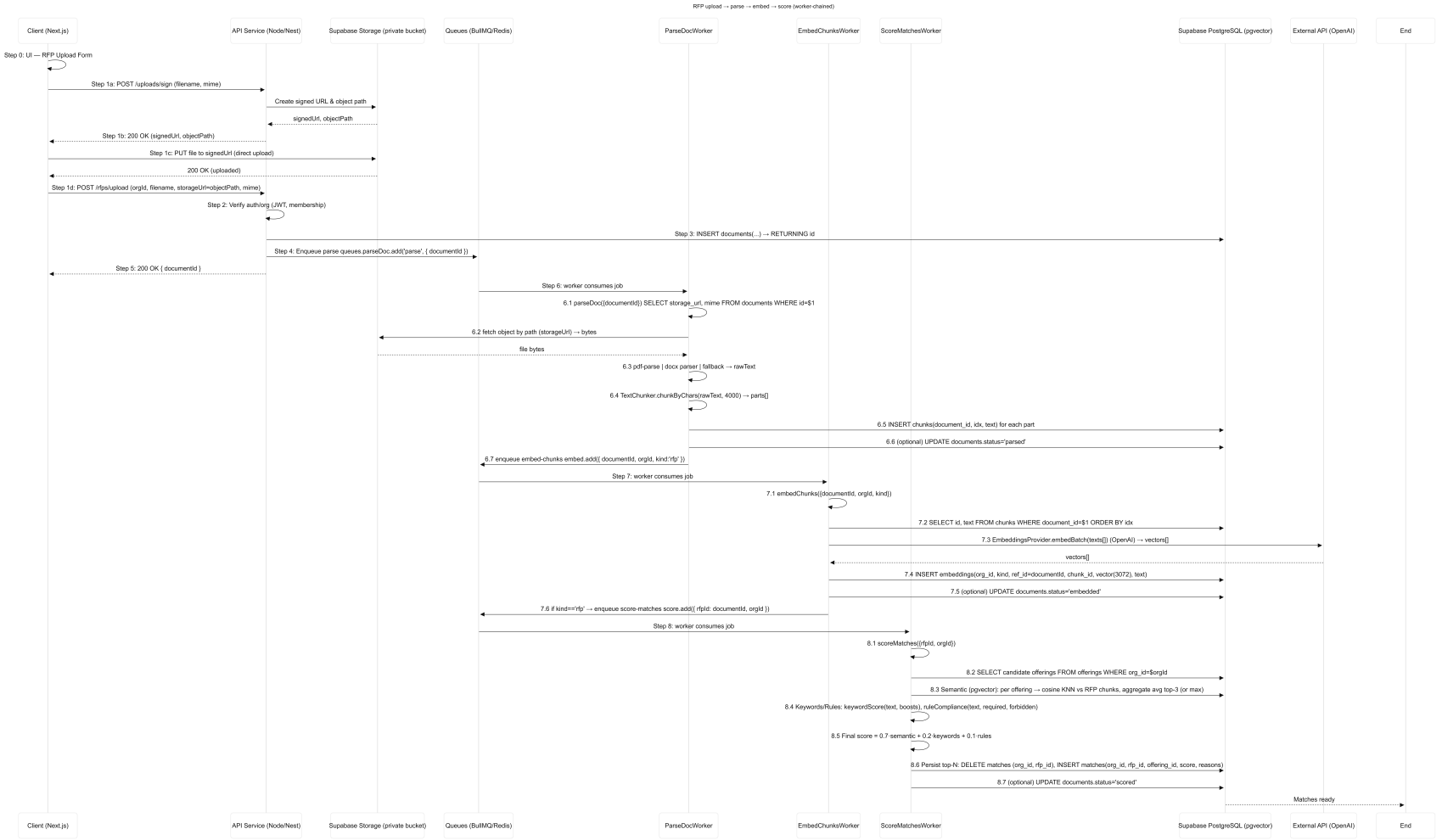


Figure 2: RFP File Upload → Parse → Embed → Score (worker-chained)

11.2.3 POST /uploads/sign — Create Signed Upload URL (Supabase Storage)

Listing 33: Express route: POST /uploads/sign (signed URL + object path)

```
// Example: return a short-lived signedUrl and an objectPath to use as storageUrl later.
app.post('/uploads/sign', async (req, res) => {
  const orgId = req.user.orgId; // enforce auth + tenant scoping
  const { filename, mime } = req.body ?? {};
  if (!filename || !mime) return res.status(400).json({ ok:false, error:'invalid_input' });

  // Example: generate a path scoped by org and date; adapt to your storage SDK.
  const objectPath = `org=${orgId}/offerings/${Date.now()}-${filename}`;
  // replace with real Supabase/S3 pre-sign call
  const signedUrl = await storage.presignPut(objectPath, { contentType: mime, expiresIn: 60 * 5 });

  return res.json({ signedUrl, objectPath });
});
```

11.2.4 POST /rfps/:rfpId/score — Force Re-Score (legacy alias)

Listing 34: Express route: POST /rfps/:rfpId/score (alias of /match)

```
// NOTE: Prefer POST /rfps/:rfpId/match (see below). This legacy alias triggers
// the same scoring job with server defaults and no overrides.
app.post('/rfps/:rfpId/score', async (req,res) => {
  const orgId = req.user.orgId;             // derive from auth; do NOT trust client
  const rfpId = req.params.rfpId;

  // Delegate to the same enqueue used by /match, with default k/topN.
  await queues.scoreMatches.add(
    'score',
    { orgId, rfpId, k: 3, topN: 10, reason: 'legacy-score' },
    { attempts: 3, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true }
  );

  res.json({ ok: true, deprecated: true, use: `/rfps/${rfpId}/match` });
});
```

11.2.5 GET /rfps — List RFPs

Returns the caller's RFPs for an organization with cursor pagination and optional search.

Query parameters.

- `limit` (optional, default 20, max 100)
- `cursor` (optional; opaque tuple encoded as `created_at|id`)
- `q` (optional; case-insensitive search over title/filename)

Listing 35: Express route: GET /rfps (cursor pagination + search)

```
// GET /rfps?limit=20&cursor=2025-09-01T10:00:00Z|f9b1...&q=soc2
app.get('/rfps', async (req, res) => {
  // NEVER trust orgId from query; derive from auth/JWT or RLS
  const orgId = req.user.orgId; // e.g., set by auth middleware

  const limit = Math.min(parseInt(String(req.query.limit ?? '20'), 10) || 20, 100);
  const cursor = typeof req.query.cursor === 'string' ? req.query.cursor : undefined;
  const q      = typeof req.query.q === 'string' ? req.query.q.trim() : '';

  // Parse cursor "created_at|id"
  let cursorCreatedAt: string | null = null;
  let cursorId: string | null = null;
  if (cursor) {
    const [ca, id] = cursor.split('|');
    cursorCreatedAt = ca || null;
    cursorId = id || null;
  }

  // Build dynamic WHERE
  const params: any[] = [orgId];
  let where = `d.org_id = $1 AND d.kind = 'rfp'`;
  if (q) {
    params.push(`%${q}%`);
    where += ` AND (d.filename ILIKE $${params.length})`;
  }

  // Cursor predicate: stable tuple (created_at DESC, id DESC)
  let cursorSql = '';
  if (cursorCreatedAt && cursorId) {
    params.push(cursorCreatedAt, cursorId);
    cursorSql = `
      AND (d.created_at, d.id) < ($${params.length-1}::timestamptz, $${params.length}::
    uuid)
    `;
  }

  params.push(limit);

```

```
  const { rows } = await pool.query(
    `
    SELECT
      d.id                AS rfp_id,
      d.filename         AS title,
      d.created_at,
      COALESCE(r.last_scored_at, NULL) AS last_scored_at,
      -- lightweight stats
      (SELECT COUNT(*)::int FROM matches m WHERE m.org_id = d.org_id AND m.rfp_id = d.id)
     AS matches_count
    FROM documents d
    LEFT JOIN rfp_meta r
      ON r.org_id = d.org_id AND r.rfp_id = d.id
    WHERE ${where}
      ${cursorSql}
    ORDER BY d.created_at DESC, d.id DESC
    LIMIT $${params.length}
    `,
    params
  );

  // Next cursor is last row's (created_at|id)
  const next =
    rows.length === limit
      ? `${rows[rows.length - 1].created_at.toISOString()}|${rows[rows.length - 1].rfp_id
    }`
      : null;

  res.json({ items: rows, nextCursor: next });
});
```

Indexes & RLS.

Listing 36: Indexes/RLS to support GET /rfps

```
-- Speed up listing + cursor ordering
CREATE INDEX IF NOT EXISTS documents_rfp_list_idx
  ON documents (org_id, kind, created_at DESC, id);

-- Optional metadata for freshness
-- rfp_meta(org_id uuid, rfp_id uuid, last_scored_at timestamptz, PRIMARY KEY (org_id,
    rfp_id))

-- Row-level security (multitenancy)
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_rfps ON documents
```

```
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid AND kind = 'rfp');
```

11.2.6 GET /rfps/:rfpId/matches — Fetch Ranked Matches

Listing 37: Express route: GET /rfps/:rfpId/matches

```
// GET /rfps/:rfpId/matches?limit=20&cursor=<offering_id>|null
app.get('/rfps/:rfpId/matches', async (req, res) => {
  const auth = req.user!;                    // derive orgId from auth
  const orgId = auth.orgId;                  // NOT from query
  const limit = Math.min(Number(req.query.limit ?? 20), 100);
  const cursor = req.query.cursor as string | undefined;

  const params: any[] = [orgId, req.params.rfpId, limit];
  const whereCursor = cursor ? `AND (m.score, m.offering_id) < (
      SELECT score, offering_id FROM matches
      WHERE org_id=$1 AND rfp_id=$2 AND offering_id=$4
    )` : ``;
  if (cursor) params.push(cursor);

  const { rows } = await pool.query(
    `
    SELECT m.offering_id,
           o.title,
           m.score,
           m.reasons,
           r.last_scored_at
    FROM matches m
    JOIN offerings o ON o.id = m.offering_id
    JOIN rfp_meta r ON r.org_id = m.org_id AND r.rfp_id = m.rfp_id
    WHERE m.org_id = $1 AND m.rfp_id = $2
      ${whereCursor}
    ORDER BY m.score DESC, m.offering_id ASC
    LIMIT $3
    `,
    params
  );

  // next cursor = last offering_id of this page (stable because of tiebreaker)
  const nextCursor = rows.length === limit ? rows[rows.length - 1].offering_id : null;
  res.json({ items: rows, nextCursor });
});
```

Indexes & RLS.

Listing 38: Indexes/RLS to support GET /rfps

```
CREATE INDEX IF NOT EXISTS matches_rfpid_rank_idx
  ON matches (org_id, rfp_id, score DESC, offering_id);

-- Optional: store when the worker last wrote results
-- rfp_meta(org_id, rfp_id, last_scored_at timestamptz)

ALTER TABLE matches ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_read_matches ON matches
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid);
```

11.2.7 POST `/rfps/:rfpId/match` — Recompute Matches Now

Triggers an immediate re-score for a single RFP. Intended for admin/ops or when configuration changes (boosts/required/forbidden) require a fresh ranking.

Listing 39: Express route: POST /rfps/:rfpId/match (idempotent enqueue)

```
// Admin-only; orgId must come from auth (e.g., req.user.orgId)
app.post('/rfps/:rfpId/match', async (req, res) => {
  const orgId = req.user.orgId; // derive from JWT/session; do NOT trust client input
  const rfpId = req.params.rfpId;

  // Optional tuning overrides (validated & bounded)
  const k     = Math.max(1, Math.min(Number(req.body?.k ?? 3), 10));      // top-k for
    semantic pooling
  const topN  = Math.max(1, Math.min(Number(req.body?.topN ?? 10), 100)); // number of
    matches to persist
  const reason = String(req.body?.reason ?? 'manual');

  // Existence check (and org scoping)
  const { rows: rfpRows } = await pool.query(
    `SELECT 1 FROM documents WHERE id=$1 AND org_id=$2 AND kind='rfp'`,
    [rfpId, orgId]
  );
  if (rfpRows.length === 0) {
    return res.status(404).json({ ok:false, error:'rfp_not_found' });
  }

  // Embeddings readiness (optional but helpful)
  const { rows: embRows } = await pool.query(
    `SELECT 1 FROM embeddings WHERE org_id=$1 AND kind='rfp' AND ref_id=$2 LIMIT 1`,
    [orgId, rfpId]
  );
  if (embRows.length === 0) {
```

```
    return res.status(409).json({ ok:false, error:'embeddings_not_ready' });
  }

  // Idempotent enqueue (one in-flight job per (org, rfp))
  const jobId = `score:${orgId}:${rfpId}`;
  await queues.scoreMatches.add(
    'score',
    { orgId, rfpId, k, topN, reason },
    {
      jobId,                    // dedupe
      attempts: 3,
      backoff: { type: 'exponential', delay: 2000 },
      removeOnComplete: true,
      removeOnFail: false
    }
  );

  return res.status(200).json({ ok:true, enqueued:true, jobId });
});
```

- **Auth & RLS.** Derive `orgId` from auth and enforce RLS on `documents`/`embeddings`/`matches`.
- **Idempotency.** The stable `jobId` prevents duplicate in-flight recomputes for the same RFP.
- **Safety.** The embeddings-readiness guard avoids enqueueing useless jobs before `embed-chunks` finishes.
- **Overrides.** Optional `k` and `topN` allow controlled rescoring; defaults match the worker's configured values.

11.3 Offering Endpoints

11.3.1 POST `/offerings` — Create Offering and Embed, Manual-Text Only

Creates an offering (title, description, tags), materializes it as a document of kind `'offering'`, inserts one chunk (`title + description`), and enqueues embeddings. The worker will write vectors to `embeddings` and the offering will participate in scoring.

Request body.

- `title` *(string, required)*
- `description` *(string, required)*
- `tags[]` *(string[], optional)*

Listing 40: Express route: POST /offerings (create + embed enqueue)

```
// Admin/user auth middleware must set req.user.orgId (do not trust client-sent orgId)
app.post('/offerings', async (req, res) => {
```

```
  const orgId = req.user.orgId;
  const { title, description, tags } = req.body ?? {};

  // Basic validation
  if (typeof title !== 'string' || !title.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_title' });
  }
  if (typeof description !== 'string' || !description.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_description' });
  }
  if (tags && !Array.isArray(tags)) {
    return res.status(400).json({ ok:false, error:'invalid_tags' });
  }

  // Create offering
  const off = await pool.query(
    `INSERT INTO offerings(org_id, title, description, tags)
     VALUES ($1, $2, $3, COALESCE($4::text[], '{}'))
     RETURNING id`,
    [orgId, title, description, tags ?? []]
  );
  const offeringId = off.rows[0].id;

  // Create document(kind='offering') linked to offering
  const doc = await pool.query(
    `INSERT INTO documents(org_id, kind, ref_id, filename, mime)
     VALUES ($1, 'offering', $2, $3, 'text/plain')
     RETURNING id`,
    [orgId, offeringId, title]
  );
  const documentId = doc.rows[0].id;

  // Insert a single chunk (title + description); we can split further if long
  await pool.query(
    `INSERT INTO chunks(document_id, idx, text)
     VALUES ($1, 0, $2)`,
    [documentId, `${title}\n\n${description}`]
  );

  // Enqueue embeddings for this offering document
  await queues.embedChunks.add(
    'embed',
    { documentId, orgId, kind: 'offering' },
    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
```

```
    }
  );

  return res.status(201).json({ ok:true, offeringId, documentId });
});
```

- **Auth/RLS.** Derive `orgId` from auth; enable RLS on `documents`, `chunks`, `embeddings`, `offerings`.
- **Chunking.** For long descriptions, prefer the same chunker as RFPs; here we keep MVP simple with one chunk.
- **Participation in scoring.** Once embeddings are written, offerings are candidates for `score-matches` and will appear in `GET /rfps/:rfpId/matches`.

DDL & Indexing.

Listing 41: Indexes and RLS policy sketches

```
-- Fast lookup by org and recency
CREATE INDEX IF NOT EXISTS offerings_org_created_idx
  ON offerings (org_id, created_at DESC);

-- Documents/chunks for offerings
CREATE INDEX IF NOT EXISTS documents_offerings_idx
  ON documents (org_id, kind, ref_id);

ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_crud_offerings ON offerings
  USING (org_id = current_setting('app.org_id', true)::uuid)
  WITH CHECK (org_id = current_setting('app.org_id', true)::uuid);
```

11.3.2 POST `/offerings/upload` — Register File-Backed Offering

Registers an offering whose source is a file in Storage (PDF/DOCX, etc.). The worker chain will parse → chunk → embed.

Request body.

- `title` *(string, required)*
- `storageUrl` *(string, required)* — S3/Supabase URL (client uploads via signed URL)
- `mime` *(string, required)* — e.g., `application/pdf`
- `tags[]` *(string[], optional)*

Listing 42: Express route: POST /offerings/upload (file-backed + parse chain)

```
// Auth middleware sets req.user.orgId; never trust client-sent orgId
```

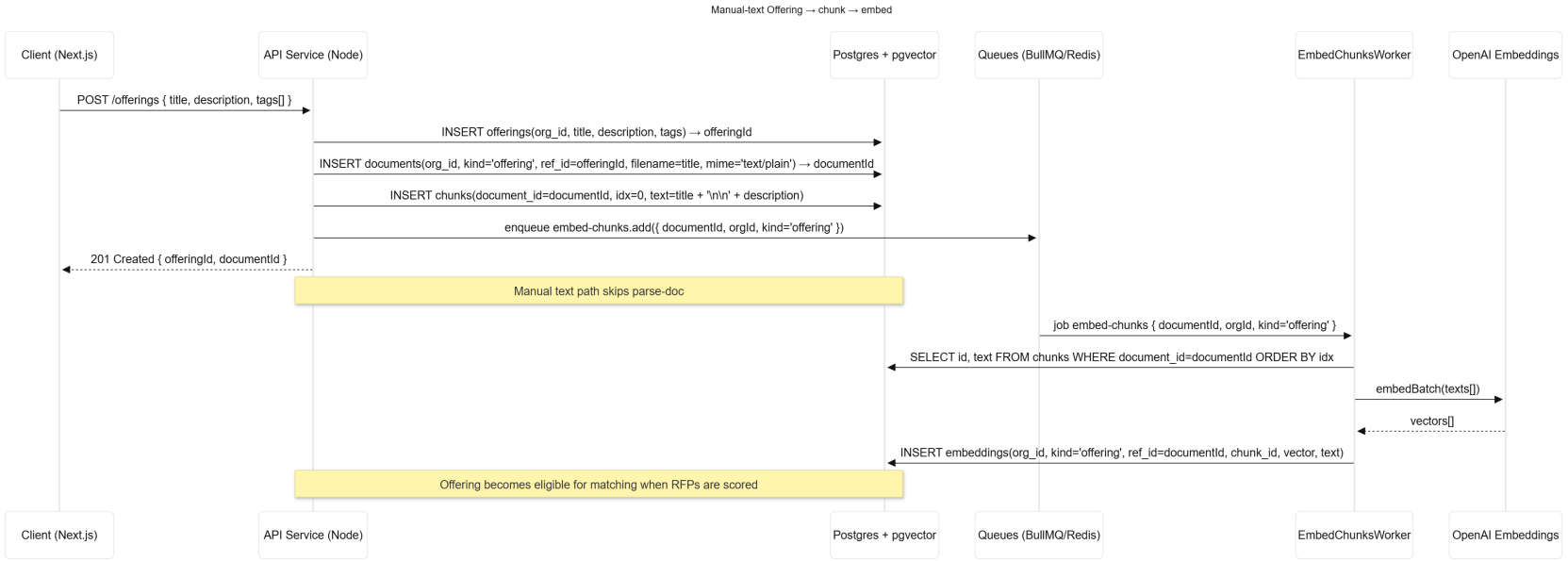


Figure 3: End-to-end sequence for `POST /offerings` (manual-text only). The API creates the offering, materializes a document and a single chunk (`title + description`), then enqueues `embed-chunks`. The worker embeds via OpenAI and persists vectors to `embeddings`. No parsing step is required; the offering becomes eligible for matching when RFPs are scored.

```
app.post('/offerings/upload', async (req, res) => {
  const orgId = req.user.orgId;
  const { title, storageUrl, mime, tags } = req.body ?? {};

  // Basic validation
  if (typeof title !== 'string' || !title.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_title' });
  }
  if (typeof storageUrl !== 'string' || !storageUrl.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_storageUrl' });
  }
  if (typeof mime !== 'string' || !mime.trim()) {
    return res.status(400).json({ ok:false, error:'invalid_mime' });
  }
  if (tags && !Array.isArray(tags)) {
    return res.status(400).json({ ok:false, error:'invalid_tags' });
  }

  // Create offering row
  const off = await pool.query(
    `INSERT INTO offerings(org_id, title, description, tags)
     VALUES ($1, $2, $3, COALESCE($4::text[], '{}'))
     RETURNING id`,
    [orgId, title, /* description */ '', tags ?? []]
  );
  const offeringId = off.rows[0].id;

  // Create document(kind='offering') pointing to the file
  const doc = await pool.query(
    `INSERT INTO documents(org_id, kind, ref_id, filename, storage_url, mime)
     VALUES ($1, 'offering', $2, $3, $4, $5)
     RETURNING id`,
    [orgId, offeringId, title, storageUrl, mime]
  );
  const documentId = doc.rows[0].id;

  // Enqueue parse → (embed) → (score-later when RFPs are scored)
  await queues.parseDoc.add(
    'parse',
    { documentId }, // parseDoc will enqueue embed-chunks
    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
    }
  );

```

```
  return res.status(201).json({ ok:true, offeringId, documentId });
});
```

- **Upload flow.** Client performs direct upload to Storage (signed URL), then calls this endpoint with the resulting `storageUrl` and `mime`.
- **Worker chain.** `parseDoc` extracts text → writes `chunks` → enqueues `embed-chunks`. No manual chunk insert here.
- **Scoring.** Offerings become candidates automatically; RFP rescoring can be manual (§11.2.7) or scheduled.

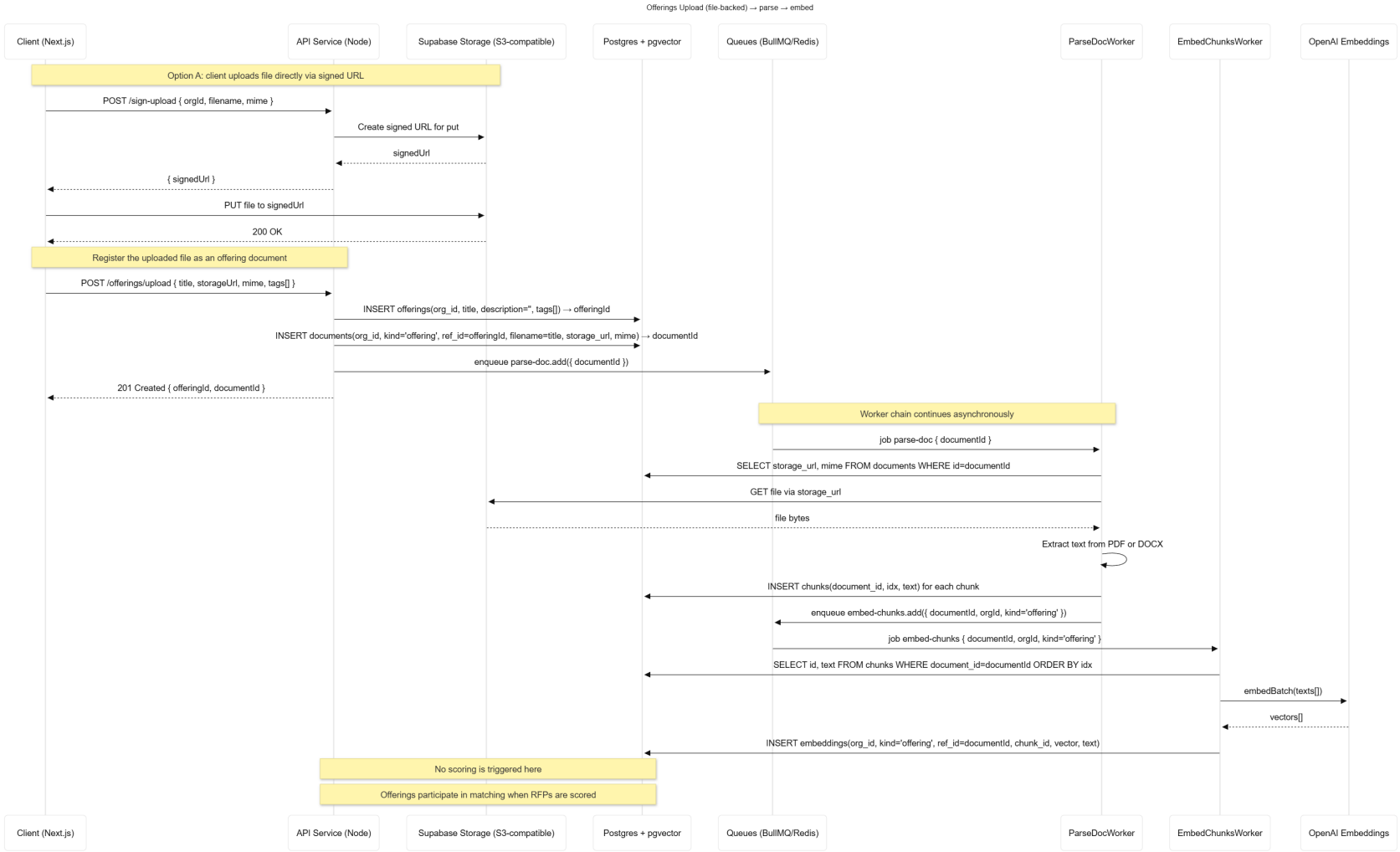


Figure 4: End-to-end sequence for `POST /offerings/upload`. The client performs a direct file upload via a signed URL to Storage; the API registers the offering and file, then enqueues `parse-doc`. The `parse-doc` worker extracts text and writes `chunks`; `embed-chunks` generates embeddings and persists them to `embeddings`. No scoring is triggered at this step; the offering becomes eligible for matching when RFPs are scored.

11.3.3 GET `/offerings` — List Offerings

Returns the caller's offerings with cursor pagination, optional search, and tag filter.

Query parameters.

- `limit` (optional, default 20, max 100)
- `cursor` (optional; opaque `created_at|id`)
- `q` (optional; case-insensitive search over `title`)
- `tag` (optional; filter where `tags` contains the value)

Listing 43: Express route: GET /offerings (cursor pagination + search + tag)

```
// GET /offerings?limit=20&cursor=2025-09-01T10:00:00Z|f9b1...&q=cloud&tag=Security
app.get('/offerings', async (req, res) => {
  const orgId = req.user.orgId; // derive from auth
  const limit  = Math.min(parseInt(String(req.query.limit ?? '20'), 10) || 20, 100);
  const cursor = typeof req.query.cursor === 'string' ? req.query.cursor : undefined;
  const q      = typeof req.query.q === 'string' ? req.query.q.trim() : '';
  const tag    = typeof req.query.tag === 'string' ? req.query.tag.trim() : '';

  // Parse cursor "created_at|id"
  let cursorCreatedAt = null, cursorId = null;
  if (cursor) [cursorCreatedAt, cursorId] = cursor.split('|');

  const params = [orgId];
  let where = `o.org_id = $1`;
  if (q)   { params.push(`%${q}%`); where += ` AND (o.title ILIKE $${params.length})`; }
  if (tag && tag !== 'All') { params.push(tag); where += ` AND ($${params.length} = ANY(o.
    tags))`; }

  let cursorSql = '';
  if (cursorCreatedAt && cursorId) {
    params.push(cursorCreatedAt, cursorId);
    cursorSql = ` AND (o.created_at, o.id) < ($${params.length-1}::timestamptz, $${params.
    length}::uuid) `;
  }
  params.push(limit);

  const { rows } = await pool.query(
    `
    WITH latest_doc AS (
      SELECT DISTINCT ON (ref_id) id, ref_id
      FROM documents
      WHERE org_id=$1 AND kind='offering'
      ORDER BY ref_id, created_at DESC
```

```
    ),
    embed_status AS (
      SELECT d.ref_id AS offering_id,
             CASE
               WHEN EXISTS (SELECT 1 FROM embeddings e WHERE e.org_id=$1 AND e.kind='
    offering' AND e.ref_id=d.id) THEN 'OK'
               ELSE 'Pending'
             END AS embed_status
      FROM latest_doc d
    )
    SELECT o.id, o.title, o.tags,
           COALESCE(es.embed_status, 'Pending') AS "embedStatus",
           o.created_at
    FROM offerings o
    LEFT JOIN embed_status es ON es.offering_id = o.id
    WHERE ${where} ${cursorSql}
    ORDER BY o.created_at DESC, o.id DESC
    LIMIT $${params.length}
    `,
    params
  );

  const next =
    rows.length === limit
      ? `${rows[rows.length - 1].created_at.toISOString()}|${rows[rows.length - 1].id}`
      : null;

  res.json({ items: rows, nextCursor: next });
});
```

Indexes & RLS.

Listing 44: Indexes/RLS for GET /offerings

```
CREATE INDEX IF NOT EXISTS offerings_org_created_idx
  ON offerings (org_id, created_at DESC, id);

ALTER TABLE offerings ENABLE ROW LEVEL SECURITY;
CREATE POLICY org_can_list_offerings ON offerings
  FOR SELECT USING (org_id = current_setting('app.org_id', true)::uuid);
```

11.3.4 GET `/offerings/:id` — Read One Offering

Listing 45: Express route: GET /offerings/:id

```
app.get('/offerings/:id', async (req, res) => {
  const orgId = req.user.orgId;
  const id = req.params.id;
  const { rows } = await pool.query(
    `SELECT id, title, description, tags
     FROM offerings
     WHERE org_id=$1 AND id=$2`,
    [orgId, id]
  );
  if (rows.length === 0) return res.status(404).json({ ok:false, error:'not_found' });
  res.json(rows[0]);
});
```

11.3.5 PATCH `/offerings/:id` — Update Offering (re-embed on change)

Partially updates an offering. If `title` or `description` change, the server re-materializes `document/chunk(0)` and enqueues *embed* so vectors stay fresh.

Request body (any subset).

- `title` *(string)*
- `description` *(string)*
- `tags[]` *(string[])*

Listing 46: Express route: PATCH /offerings/:id (with embed re-enqueue)

```
// Auth middleware sets req.user.orgId
app.patch('/offerings/:id', async (req, res) => {
  const orgId = req.user.orgId;
  const id = req.params.id;
  const { title, description, tags } = req.body ?? {};

  // Load current to detect changes
  const cur = await pool.query(
    `SELECT id, title, description, tags FROM offerings WHERE org_id=$1 AND id=$2`,
    [orgId, id]
  );
  if (cur.rows.length === 0) return res.status(404).json({ ok:false, error:'not_found' })
    ;

  // Build dynamic update
  const sets: string[] = [];
  const vals: any[] = [orgId, id];
  if (typeof title === 'string') { vals.push(title); sets.push(`title = $${vals.length}`)
    ; }
```

```
  if (typeof description === 'string') { vals.push(description); sets.push(`description =
     $${vals.length}`); }
  if (Array.isArray(tags)) { vals.push(tags); sets.push(`tags = COALESCE($${vals.length
    }::text[], '{}')`); }

  if (sets.length === 0) return res.json({ ok:true, updated:false });

  await pool.query(
    `UPDATE offerings SET ${sets.join(', ')}, updated_at = NOW()
     WHERE org_id = $1 AND id = $2`,
    vals
  );

  // If text changed, re-materialize document/chunk(0) and re-embed
  const textChanged = (typeof title === 'string') || (typeof description === 'string');
  if (textChanged) {
    // Find or create the 'offering' document for this offering
    const doc = await pool.query(
      `SELECT id FROM documents WHERE org_id=$1 AND kind='offering' AND ref_id=$2 LIMIT
    1`,
      [orgId, id]
    );
    let documentId: string;
    if (doc.rows.length === 0) {
      const created = await pool.query(
        `INSERT INTO documents(org_id, kind, ref_id, filename, mime)
         VALUES ($1, 'offering', $2, $3, 'text/plain')
         RETURNING id`,
        [orgId, id, title ?? cur.rows[0].title]
      );
      documentId = created.rows[0].id;
    } else {
      documentId = doc.rows[0].id;
    }

    // Replace chunk(0) with latest title+description
    const t = (typeof title === 'string') ? title : cur.rows[0].title;
    const d = (typeof description === 'string') ? description : cur.rows[0].description;
    await pool.query(`DELETE FROM chunks WHERE document_id=$1`, [documentId]);
    await pool.query(
      `INSERT INTO chunks(document_id, idx, text) VALUES ($1, 0, $2)`,
      [documentId, `${t}\n\n${d || ''}`]
    );

```

```
    // Enqueue embeddings
    await queues.embedChunks.add(
      'embed',
      { documentId, orgId, kind: 'offering' },
      { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete:
    true }
    );
  }

  res.json({ ok:true, updated:true, reembedded: textChanged });
});
```

Remarks.

- **Security**: `orgId` is derived from auth; RLS is enforced on all tables (`offerings`, `documents`, `chunks`, `embeddings`).
- **Idempotency**: updating with no text changes skips embed re-enqueue for efficiency.

11.3.6 POST `/offerings/:id/reembed` — Force Re-embed

Enqueues an embedding job for the offering's current text without requiring edits. Useful for model/version changes or after external data updates.

Listing 47: Express route: POST /offerings/:id/reembed

```
app.post('/offerings/:id/reembed', async (req, res) => {
  const orgId = req.user.orgId;
  const id = req.params.id;

  // Find offering document
  const d = await pool.query(
    `SELECT id FROM documents WHERE org_id=$1 AND kind='offering' AND ref_id=$2 LIMIT 1`,
    [orgId, id]
  );
  if (d.rows.length === 0) {
    return res.status(404).json({ ok:false, error:'document_not_found' });
  }
  const documentId = d.rows[0].id;

  await queues.embedChunks.add(
    'embed',
    { documentId, orgId, kind: 'offering', reason: 'manual-reembed' },
    { attempts: 3, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true
    }
  );

```

```
  res.json({ ok:true, enqueued:true });
});
```

Remarks.

- **Security**: `orgId` derived from auth; enforce RLS across `offerings`, `documents`, `chunks`, `embeddings`.
- **Operational clarity**: `PATCH` covers normal edits (auto re-embed on text change). `POST /reembed` provides an explicit tool for ops without mutating content.

11.4 KB Endpoints

11.4.1 POST `/kb/upload` — Register File-Backed KB

Registers a KB document already uploaded to Storage (client-side signed URL flow). Enqueues parse → embed; rows are RLS-scoped to the caller's org.

Request body.

- `title` *(string, required)*
- `storageUrl` *(string, required)* — S3/Supabase URL
- `mime` *(string, required)* — e.g., `application/pdf`

Note: `orgId` is derived from auth and applied via RLS (do not accept from client).

Listing 48: Express route: POST /kb/upload

```
app.post('/kb/upload', requireAuth, withPgScope, async (req, res) => {
  const { title, storageUrl, mime } = req.body ?? {};
  if (typeof title !== 'string' || !title.trim())       return res.status(400).json({
    error:'invalid_title' });
  if (typeof storageUrl !== 'string' || !storageUrl)    return res.status(400).json({
    error:'invalid_storageUrl' });
  if (typeof mime !== 'string' || !mime.trim())         return res.status(400).json({
    error:'invalid_mime' });

  // Materialize a KB document (no orgId in SQL; RLS handles tenancy)
  const doc = await (req as any).pg.query(`
    INSERT INTO documents(kind, filename, storage_url, mime)
    VALUES ('kb', $1, $2, $3)
    RETURNING id
  `, [title, storageUrl, mime]);

  const documentId = doc.rows[0].id;

  // Enqueue parse -> (embed) via worker chain
  await queues.parseDoc.add(
```

```
    'parse',
    { documentId },
    { attempts: 5, backoff: { type: 'exponential', delay: 2000 }, removeOnComplete: true }
  );

  return res.status(201).json({ ok: true, documentId });
});
```

Remarks.

- **Upload flow.** Use client direct upload (signed URL), then call this endpoint with the resulting `storageUrl`.
- **Worker chain.** `parse-doc` extracts text → writes `chunks` → enqueues `embed-chunks`.
- **Assistant.** KB embeddings (`kind='kb'`) are included in retrieval for `/assistant/draft`.

11.4.2 GET `/kb` — List KB Documents (filters for Kind/Status)

Lists KB documents for the caller's org (RLS-scoped). Supports search, pagination, and UI-aligned filters for document *Kind* (PDF/DOCX) and pipeline *Status* (Parsed/Embedded/Pending).

Query params.

- `q` *(string, optional)* — case-insensitive substring match on `filename`.
- `kind` *(string, optional; one of `All|PDF|DOCX`)* — maps to MIME family.
- `status` *(string, optional; one of `All|Parsed|Embedded|Pending`)* — computed from pipeline progress.
- `limit` *(int, optional, default 20, max 100)*
- `offset` *(int, optional, default 0)*

Listing 49: Express route: GET /kb (list with kind/status filters)

```
app.get('/kb', requireAuth, withPgScope, async (req, res) => {
  const q       = (req.query.q as string | undefined)?.trim() || '';
  const kind    = (req.query.kind as string | undefined)?.trim() || 'All';       // All | PDF | DOCX
  const status  = (req.query.status as string | undefined)?.trim() || 'All';     // All | Parsed | Embedded | Pending
  const limit   = Math.min(Math.max(parseInt((req.query.limit as string) || '20', 10), 1), 100);
  const offset  = Math.max(parseInt((req.query.offset as string) || '0', 10), 0);

  // Build WHERE for documents(kind='kb'), plus optional filename + mime-family filters
  const params: any[] = [];
  let where = `d.kind = 'kb'`;

```

```
12  if (q) {
13    params.push(`%${q.toLowerCase()}%`);
14    where += ` AND lower(d.filename) LIKE $${params.length}`;
15  }
16
17  // Map "Kind" -> MIME predicates
18  if (kind === 'PDF') {
19    where += ` AND (d.mime ILIKE 'application/pdf' OR d.mime ILIKE '%pdf%')`;
20  } else if (kind === 'DOCX') {
21    where += ` AND (d.mime IN (
22                 'application/vnd.openxmlformats-officedocument.wordprocessingml.
      document',
23                 'application/msword'
24               ) OR d.mime ILIKE '%word%')`;
25  }
26  // else "All" -> no extra filter
27
28  // Status filter is applied after we compute pipeline status in a CTE
29  const statusFilterSql =
30    status === 'Embedded' ? `WHERE p.status = 'Embedded'` :
31    status === 'Parsed'   ? `WHERE p.status = 'Parsed'`   :
32    status === 'Pending'  ? `WHERE p.status = 'Pending'`  :
33                            ``; // All
34
35  params.push(limit, offset);
36
37  const { rows } = await (req as any).pg.query(
38    `
39    WITH doc AS (
40      SELECT d.id, d.filename, d.mime, d.created_at
41      FROM documents d
42      WHERE ${where}
43      ORDER BY d.created_at DESC
44      LIMIT $${params.length - 1} OFFSET $${params.length}
45    ),
46    pipeline AS (
47      SELECT
48        doc.id AS document_id,
49        CASE
50          WHEN EXISTS (
51            SELECT 1 FROM embeddings e
52            WHERE e.kind = 'kb' AND e.ref_id = doc.id
53            LIMIT 1
54          ) THEN 'Embedded'
```

```
          WHEN EXISTS (
            SELECT 1 FROM chunks c
            WHERE c.document_id = doc.id
            LIMIT 1
          ) THEN 'Parsed'
          ELSE 'Pending'
        END AS status
      FROM doc
    )
    SELECT doc.id AS document_id,
           doc.filename,
           doc.mime,
           doc.created_at,
           p.status
    FROM doc
    JOIN pipeline p ON p.document_id = doc.id
    ${statusFilterSql}
    ORDER BY doc.created_at DESC
    `,
    params
  );

  res.json({ items: rows, limit, offset });
});
```

Remarks.

- **Kind filter.** The UI's *Kind* (`All|PDF|DOCX`) maps to MIME-family checks; extend as needed for more types.
- **Status filter.** *Embedded* means at least one row in `embeddings(kind='kb', ref_id=document_id)`; *Parsed* means at least one row in `chunks(document_id)`; otherwise *Pending*.
- **RLS.** The `withPgScope` middleware (or equivalent) ensures `current_setting('app.org_id')` is set so RLS policies scope results to the caller's org without passing `orgId` from the client.

11.5 Assistant Endpoints

11.5.1 POST `/assistant/draft` — RAG Draft

Purpose.

Entry point for the drafting assistant that turns RFP context into draft prose.

Listing 50: Express route: POST /assistant/draft (orgId derived via RLS)

```
// apps/api/src/server.ts
import { requireAuth } from './auth';
import { withPgScope } from './pg-scope';
```

```
import { draftAssistant } from './svc.assistant';

app.post('/assistant/draft', requireAuth, withPgScope, async (req, res) => {
  // orgId is derived from JWT by requireAuth and applied to the PG session by
    withPgScope
  const { rfpId, question } = req.body || {};
  if (!rfpId || !question) {
    return res.status(400).json({ error: 'rfpId_and_question_required' });
  }

  const out = await draftAssistant({
    rfpId,
    question,
    client: (req as any).pg // request-scoped PG client with SET LOCAL app.org_id
  });

  res.json(out); // { text, sources }
});
```

Request/Response.

- **Request**: `{ rfpId: UUID, question: string }`
- **Response**: `{ text: string, sources: string[] }`

11.6 API–Worker Bridge (`apps/api/worker-bridge.ts`)

Listing 51: Expose queues to API layer

```
import { makeQueue, defaultJobOpts } from '../worker/src/queue';
export const queues = {
  parseDoc: makeQueue('parse-doc').q,
  embedChunks: makeQueue('embed-chunks').q,
  scoreMatches: makeQueue('score-matches').q,
};
export { defaultJobOpts };
```

12 Assistant (RAG) for Drafting Responses

This service generates proposal drafts grounded on retrieved context (RFP & KB chunks). It reuses embeddings produced elsewhere but does *not* compute or persist match scores (see §9).

Overview.

- **Context retrieval.** For the caller's organization and target RFP, retrieve the most relevant chunks/snippets (optionally including KB).

- **Prompt construction.** Assemble a prompt using the retrieved context and the user's question (e.g., "Draft Security & Compliance").
- **Generation.** Call the LLM to produce a grounded draft based on the provided context.
- **Return.** Respond with the draft text and lightweight source hints for traceability.

Usage.

- Draft proposal sections (e.g., "Company Overview", "SLA/Uptime", "Compliance").
- Answer specific RFP questions in the organization's house style.
- Produce cover letters, executive summaries, or clarifications.

Behavioral remarks.

- **Read-only to scoring.** This endpoint does not alter matches or scores; it only reads context and generates text.
- **Grounding discipline.** The prompt instructs the model to answer strictly from retrieved context and to call out missing information.
- **Latency/limits.** Keep a small top-k context (e.g., $k \in [6, 12]$), deduplicate near-duplicates, and trim long chunks to control latency and token usage.
- **Auth/RLS.** Enforce tenant scoping via authentication-derived `orgId` and Postgres RLS on `embeddings`/`documents`.

12.1 Service Implementation (`apps/api/src/svc.assistant.ts`)

Listing 52: Retrieve top chunks (RFP + KB) semantically and call OpenAI

```
// apps/api/src/svc.assistant.ts
import OpenAI from 'openai';
import type { PoolClient } from 'pg';

const oai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });

/**
 * NOTE:
 *  - `orgId` is NOT accepted from the client. Tenant scoping is enforced by Postgres RLS.
 *  - Use the request-scoped `client` (middleware sets `SET LOCAL app.org_id = ...`).
 */
export async function draftAssistant({
  rfpId,
  question,
  client
}: {
  rfpId: string;
  question: string;
```

```
  client: PoolClient;   // request-scoped client with RLS org set
}) {
  // Embed the question once
  const emb = await oai.embeddings.create({
    model: 'text-embedding-3-large',
    input: question
  });
  const qvec = `[${emb.data[0].embedding.join(',')}]`;

  // Semantic KNN over this RFP + org KB (RLS limits rows to caller's org)
  const { rows: hits } = await client.query(
    `
    WITH cand AS (
      SELECT kind, ref_id, text, vector
      FROM embeddings
      WHERE (kind = 'rfp' AND ref_id = $1) OR kind = 'kb'
    )
    SELECT kind,
           ref_id,
           text,
           1 - (vector <=> $2::vector) AS sim
    FROM cand
    ORDER BY vector <=> $2::vector      -- cosine distance asc
    LIMIT 12
    `,
    [rfpId, qvec]
  );

  // Light dedupe + clamp to token budget
  const seen = new Set<string>();
  const top: Array<{ kind: string; ref_id: string; text: string; sim: number }> = [];
  for (const h of hits) {
    const text = (h.text || '') as string;
    const key = text.slice(0, 160).toLowerCase(); // cheap near-dup heuristic
    if (seen.has(key)) continue;
    seen.add(key);
    top.push({ kind: h.kind, ref_id: h.ref_id, text, sim: Number(h.sim) });
    if (top.length >= 8) break;
  }
  const ctxTexts = top.map(h => h.text.slice(0, 1200)); // trim long chunks

  // Prompt
  const messages = [
    {
```

```
      role: 'system' as const,
      content:
        'You are a proposal assistant. Answer strictly using the provided CONTEXT. ' +
        'If information is missing, state what is needed.'
    },
    {
      role: 'user' as const,
      content: `CONTEXT:\n${ctxTexts.join('\n---\n')}\n\nQUESTION:\n${question}`
    }
  ];

  // Call LLM
  const resp = await oai.chat.completions.create({
    model: 'gpt-4o-mini',
    messages,
    temperature: 0.2
  });

  // Return draft + useful citations
  return {
    text: resp.choices[0]?.message?.content ?? '',
    sources: top.map((h, i) => ({
      rank: i + 1,
      kind: h.kind,
      refId: h.ref_id,
      sim: h.sim
    }))
  };
}
```

13 Frontend Integration (Next.js)

This section shows minimal client flows that call the API endpoints defined in Section 11. The pages demonstrate:

- Creating a manual-text RFP and polling for matches.
- Asking the assistant to draft content grounded on retrieved chunks.

13.1 Login Page (`apps/web/app/login/page.tsx`)

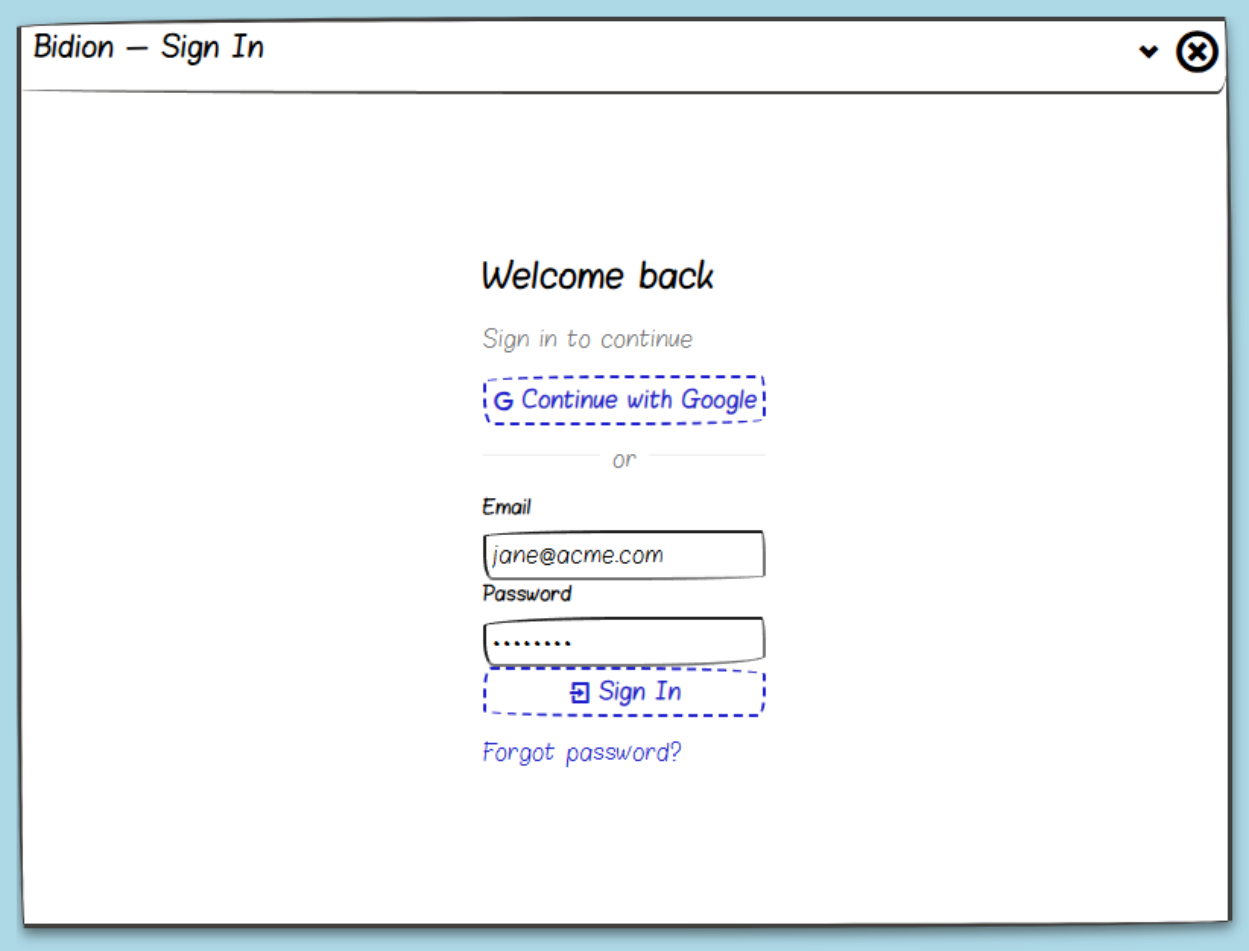


Figure 5: Login screen (email/password & Google).

Goal: present a minimal sign-in form (email + password) and a Google SSO button using **Supabase Auth** (we lock in Supabase Auth with the **Google** provider enabled), persist a session, and navigate to the dashboard.

Platform setup (Supabase Auth + Google).

1. In Supabase Dashboard: *Authentication* → *Providers* → *Google* → Enable. Provide the Google OAuth Client ID/Secret. Set redirect URL(s) to your Next.js domain(s), e.g. `http://localhost:3000/auth/c` and production URL(s).
2. In your web app env, set:
   - `NEXT_PUBLIC_SUPABASE_URL=...`
   - `NEXT_PUBLIC_SUPABASE_ANON_KEY=...`
3. Server routes (optional, production): use Next.js server actions or API routes to issue HTTP-only cookies via `@supabase/auth-helpers-nextjs`.

UX/Behavior.

- Fields: email, password; primary button: *Sign In*; secondary: *Sign in with Google*; link: *Forgot password?*
- On submit (locked-in path): call `supabase.auth.signInWithPassword({ email, password })`; handle `error` or `data.session`.
- Google SSO: call `supabase.auth.signInWithOAuth({ provider: 'google', options: { redirectTo } })`.
- Session persistence: MVP can store the access token client-side; production should set HTTP-only cookies via a server handler.
- Legacy (alternative) path retained below: POST to `/auth/login` with JSON `{ email, password }` (if keeping a custom backend).

Listing 53: Supabase Auth (email/password & Google) – client-side

```
'use client';
import { useState } from 'react';
import { useRouter } from 'next/navigation';
import { createClient } from '@supabase/supabase-js';

const supabase = createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
);

export default function LoginPage() {
  const [email, setEmail] = useState('jane@acme.com');
  const [password, setPassword] = useState('');
  const [busy, setBusy] = useState(false);
  const [err, setErr] = useState('');
```

```
  const router = useRouter();

  async function signInEmail() {
    setBusy(true); setErr('');
    const { data, error } = await supabase.auth.signInWithPassword({ email, password });
    setBusy(false);
    if (error) { setErr(error.message); return; }
    // MVP: persist access_token; production: prefer HTTP-only cookies via server helpers
    const accessToken = data.session?.access_token;
    if (accessToken) localStorage.setItem('accessToken', accessToken);
    router.push('/dashboard');
  }

  async function signInGoogle() {
    const { error } = await supabase.auth.signInWithOAuth({
      provider: 'google',
      options: { redirectTo: `${window.location.origin}/auth/callback` }
    });
    if (error) setErr(error.message);
    // After redirect/callback, Supabase will restore the session; route guard moves user
     to /dashboard.
  }

  return (
    <div style={{ maxWidth: 420, margin: '80px auto' }}>
      <h2>Welcome back</h2>
      <p style={{ color: '#777' }}>Sign in to continue</p>
      <div style={{ display:'grid', gap:8, marginTop:12 }}>
        <label>
          <div>Email</div>
          <input type="email" value={email} onChange={e=>setEmail(e.target.value)} style
    ={{ width:'100%', padding:8 }}/>
        </label>
        <label>
          <div>Password</div>
          <input type="password" value={password} onChange={e=>setPassword(e.target.value
    )} style={{ width:'100%', padding:8 }}/>
        </label>
        <button onClick={signInEmail} disabled={busy} style={{ padding:'8px 12px' }}>
          {busy ? 'Signing in...' : 'Sign In'}
        </button>
        <button onClick={signInGoogle} style={{ padding:'8px 12px' }}>
          Continue with Google
        </button>
```

```
        <p><a href="/forgot">Forgot password?</a></p>
        {err && <p style={{ color: 'crimson' }}>{err}</p>}
      </div>
    </div>
  );
}
```

Listing 54: (Alternative) Custom backend login retained for reference

```
'use client';
import { useState } from 'react';
import { useRouter } from 'next/navigation';

export default function LegacyLogin() {
  const [email, setEmail] = useState('jane@acme.com');
  const [password, setPassword] = useState('');
  const [busy, setBusy] = useState(false);
  const [err, setErr] = useState('');
  const router = useRouter();

  async function signIn() {
    setBusy(true); setErr('');
    try {
      const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + '/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password })
      });
      if (!r.ok) throw new Error(`Login failed: ${r.status}`);
      const { accessToken, orgId } = await r.json();
      localStorage.setItem('accessToken', accessToken);
      localStorage.setItem('orgId', orgId);
      router.push('/dashboard');
    } catch (e:any) {
      setErr(e.message || 'Login error');
    } finally {
      setBusy(false);
    }
  }

  return <button onClick={signIn} disabled={busy}>{busy ? 'Signing in...' : 'Sign In (
    legacy)'}</button>;
}
```

Notes (MVP vs production).

- **Locked-in auth:** Supabase Auth with Google provider enabled. Email/password flows use `signInWithPassword`; Google uses `signInWithOAuth`.
- **Token storage:** MVP may use `localStorage`; production should use HTTP-only cookies (e.g., `@supabase/auth-helpers-nextjs`) to mitigate XSS and enable SSR protection.
- **Redirects/guards:** protect pages (e.g., `/dashboard`) and restore session on load; if no session, redirect to `/login`.

13.2 Dashboard Page (`apps/web/app/page.tsx`)

Integration map (wireframe ↔ API).

- **Overview KPIs** (*RFPs Total, Due Soon (7d), In Review, Submitted*) → `GET /dashboard/summary` (server derives `orgId` from Supabase Auth).
- **RFPs Due Soon** (*title + due date list*) → `GET /dashboard/summary` → `dueSoon[]` slice.
- **Top Matches** (*RFP title + best score teaser*) → `GET /dashboard/summary` → `topMatches[]` slice (pre-joined with best offering/score).
- **Auth headers**: include `Authorization: Bearer <access_token>` or rely on HTTP-only cookies when proxied via Next.js server routes.

Listing 55: Dashboard client: fetch summary and render KPIs/lists

```
'use client';
import { useEffect, useState } from 'react';

export default function DashboardPage() {
  const [summary, setSummary] = useState({
    totals: { rfps: 0, dueSoon7d: 0, inReview: 0, submitted: 0 },
    dueSoon: [],
    topMatches: []
  });

  async function getAccessToken() {
    // For production, prefer a Next.js route that sets an HTTP-only cookie.
    return localStorage.getItem('accessToken') || '';
  }

  useEffect(() => {
    (async () => {
      const token = await getAccessToken();
      const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + '/dashboard/summary',
    {
        headers: { 'Authorization': token ? `Bearer ${token}` : '' }
      });
```

```
      if (!r.ok) throw new Error(`summary failed: ${r.status}`);
      const json = await r.json();
      setSummary(json);
    })();
  }, []);

  return (
    <div style={{maxWidth: 1040, margin: '2rem auto'}}>
      <h2>Overview</h2>
      <div style={{display:'grid', gridTemplateColumns:'repeat(4, 1fr)', gap:12}}>
        <Card title="RFPs (Total)" value={summary.totals.rfps} />
        <Card title="Due Soon (7d)" value={summary.totals.dueSoon7d} />
        <Card title="In Review" value={summary.totals.inReview} />
        <Card title="Submitted" value={summary.totals.submitted} />
      </div>

      <section style={{marginTop:16}}>
        <h3>RFPs Due Soon</h3>
        {summary.dueSoon.map(x => (
          <div key={x.rfp_id} style={{display:'flex', justifyContent:'space-between',
    borderBottom:'1px dotted #ddd', padding:'8px 0'}}>
            <span>{x.title}</span>
            <span>Due: {new Date(x.due_date).toLocaleDateString()}</span>
          </div>
        ))}
      </section>

      <section style={{marginTop:16}}>
        <h3>Top Matches</h3>
        {summary.topMatches.map(x => (
          <div key={x.rfp_id} style={{display:'flex', justifyContent:'space-between',
    borderBottom:'1px dotted #ddd', padding:'8px 0'}}>
            <span><strong>RFP:</strong> {x.title}</span>
            <span style={{border:'1px solid #ddd', borderRadius:999, padding:'2px 8px'}}>
    Score {x.top_score.toFixed(2)}</span>
          </div>
        ))}
      </section>
    </div>
  );
}

function Card({ title, value }) {
  return (
```

```
    <div style={{border:'1px dashed #bbb', background:'#fff', borderRadius:8, padding
    :12}}>
      <h3 style={{margin:0, fontSize:14}}>{title}</h3>
      <h1 style={{margin:'6px 0 0 0'}}>{value}</h1>
    </div>
  );
}
```

Remarks (data and auth).

- **Single call**: Dashboard composes KPIs + short lists from one server-side query to reduce client round trips.
- **Org scoping**: Server derives `orgId` from Supabase Auth and enforces RLS; the client never sends `orgId`.
- **Optional schema**: Uses `rfp_meta(due_date, status, last_scored_at)` to compute KPIs and "due soon."

13.3 RFP List Page (`apps/web/app/rfps/page.tsx`)

Integration map (wireframe ↔ API).

- **Initial load / pagination** → `GET /rfps?limit={n}&cursor={...}` (see Section 11.2.5). The API derives `orgId` from auth; do *not* send it in the URL.
- **Search box** → `GET /rfps?q={term}` (server filters by filename/title). Debounce client-side to reduce calls.
- **"Open" row action** → route to `/rfps/{rfpId}` (detail page uses `GET /rfps/:rfpId` if implemented, otherwise the Matches tab can call `GET /rfps/:rfpId/matches` directly).
- **"New RFP" button** → navigate to `/rfps/new` (see Section 13.4 for create flow).
- **Status/Source filters (UI)**: current API supports `q`. Status/Source can be client-side filters for now; optionally extend `GET /rfps` later with `status`, `source` if needed.
- **Auth headers**: Supabase Auth bearer or HTTP-only cookies via a Next.js route (preferred in prod).

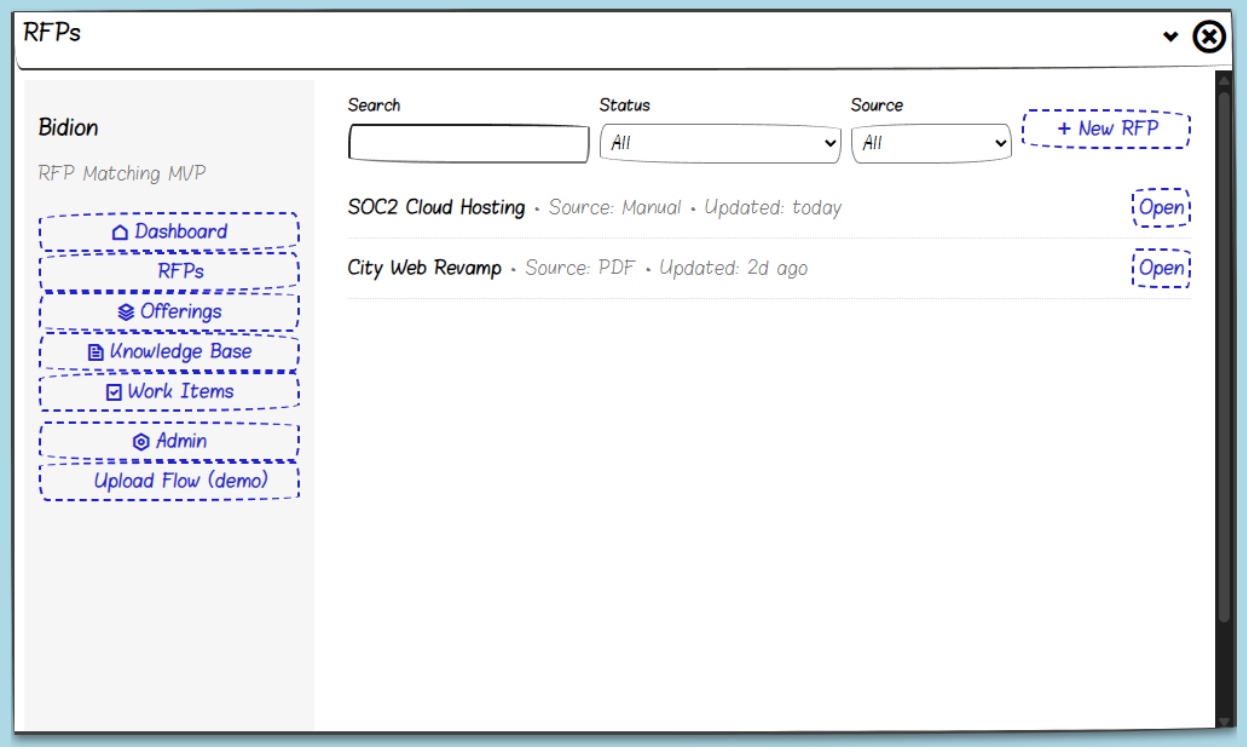


Figure 6: RFP list with search, quick actions, and create/import buttons.

Listing 56: RFP list with search + cursor pagination (aligned with GET /rfps)

```
'use client';
import { useEffect, useMemo, useState } from 'react';

export default function RfpsPage() {
  const [items, setItems] = useState([]);
  const [nextCursor, setNext] = useState(null);
  const [q, setQ] = useState('');
  const [loading, setLoading] = useState(false);
  const [err, setErr] = useState('');

  async function getAccessToken() {
    // In production, prefer HTTP-only cookies set by a Next.js route.
    return localStorage.getItem('accessToken') || '';
  }

  async function fetchPage(cursor) {
    setLoading(true); setErr('');
    try {
      const token = await getAccessToken();
      const params = new URLSearchParams();
      params.set('limit', '20');
```

```
      if (q) params.set('q', q);
      if (cursor) params.set('cursor', cursor);
      const url = process.env.NEXT_PUBLIC_API_BASE_URL + '/rfps?' + params.toString();

      const r = await fetch(url, {
        headers: { 'Authorization': token ? `Bearer ${token}` : '' }
      });
      if (!r.ok) throw new Error(`List failed: ${r.status}`);
      const json = await r.json(); // { items, nextCursor }
      // Reset list on new search; append on pagination
      if (cursor) {
        setItems(prev => [...prev, ...(json.items || [])]);
      } else {
        setItems(json.items || []);
      }
      setNext(json.nextCursor || null);
    } catch (e) {
      setErr(e instanceof Error ? e.message : 'Fetch error');
    } finally {
      setLoading(false);
    }
  }

  // Debounce search
  const debouncedQ = useDebounce(q, 300);
  useEffect(() => { fetchPage(null); /* initial + on search change */ }, [debouncedQ]);

  return (
    <div style={{maxWidth: 960, margin: '2rem auto'}}>
      <h1>RFPs</h1>
      <div style={{display:'grid', gridTemplateColumns:'1fr 200px 200px 160px', gap:8,
    alignItems:'center'}}>
        <label>
          <div>Search</div>
          <input value={q} onChange={e=>setQ(e.target.value)} placeholder="Find RFPs..."
    style={{width:'100%', padding:8}} />
        </label>
        <label>
          <div>Status</div>
          <select defaultValue="All" disabled /* API filter optional */>
            <option>All</option><option>New</option><option>Reviewing</option><option>
    Drafting</option><option>Submitted</option>
          </select>
        </label>
```

```
        <label>
          <div>Source</div>
          <select defaultValue="All" disabled /* API filter optional */>
            <option>All</option><option>Manual</option><option>CSV</option><option>API</option><option>PDF</option>
          </select>
        </label>
        <div style={{textAlign:'right'}}>
          <a href="/rfps/new" style={{display:'inline-block', padding:'8px 12px', border:'1px solid #ddd', borderRadius:8}}>New RFP</a>
        </div>
      </div>

      <div style={{marginTop:16}}>
        {items.map((x) => (
          <div key={x.rfp_id} style={{display:'flex', justifyContent:'space-between', borderBottom:'1px dotted #ddd', padding:'10px 0'}}>
            <span>
              <strong>{x.title}</strong>
              <span style={{color:'#777'}}>  Updated: {fmtDate(x.created_at)}  Matches: {x.matches_count ?? 0}</span>
            </span>
            <div>
              <a href={`/rfps/${x.rfp_id}`} style={{padding:'6px 10px', border:'1px solid #ddd', borderRadius:8}}>Open</a>
            </div>
          </div>
        ))}
        {loading && <p style={{color:'#777'}}>Loading…</p>}
        {err && <p style={{color:'crimson'}}>{err}</p>}
        {nextCursor && !loading && (
          <button onClick={()=>fetchPage(nextCursor)} style={{marginTop:12, padding:'8px 12px'}}>Load more</button>
        )}
      </div>
    </div>
  );
}

function useDebounce(value, ms) {
  const [v, setV] = useState(value);
  useEffect(() => {
    const t = setTimeout(() => setV(value), ms);
    return () => clearTimeout(t);
```

```
  }, [value, ms]);
  return v;
}

function fmtDate(iso) {
  try { return new Date(iso).toLocaleDateString(); } catch { return ''; }
}
```

Remarks.

- **Alignment:** The page uses `GET /rfps` for list/search/pagination exactly as defined in Section 11.2.5; `orgId` is derived from auth (RLS).
- **Filters:** The wireframe shows *Status* and *Source*. Until the API adds these fields, the controls are rendered but disabled. If later required, extend `GET /rfps` with `status` and `source` query params and matching indexes.
- **Navigation:** The *Open* action routes to `/rfps/{rfpId}`. That page can call `GET /rfps/:rfpId` (if implemented) and/or `GET /rfps/:rfpId/matches`.
- **Auth:** In production, prefer issuing HTTP-only cookies from a Next.js server route that exchanges the Supabase session for a backend session; the client example uses a bearer token for clarity.

13.4 RFP Creation Page (`apps/web/app/rfps/new/page.tsx`)

Integration map (wireframe ↔ API).

- **Manual text** → `POST /rfps` with `{ title, bodyText }`; *do not* send `orgId` (server derives it from Supabase Auth). Worker chain continues with `embed-chunks`. See Section 11.2.1.
- **File-backed (PDF/DOCX)**:
  1. `POST /uploads/sign` with `{ filename, mime }` → returns `signedUrl`, `objectPath`.
  2. `PUT` file bytes to `signedUrl` (direct to Supabase Storage).
  3. `POST /rfps/upload` with `{ filename, storageUrl=objectPath, mime }`; *do not* send `orgId` (server derives it). Worker chain begins with `parse-doc`. See Section 11.2.2.
- **Refresh Matches** → `GET /rfps/:rfpId/matches`; derive `orgId` from the authenticated request (RLS); do *not* pass it in the query.
- **Auth headers**: with Supabase Auth, include `Authorization: Bearer <access_token>` (or rely on HTTP-only cookies via Next.js server routes).

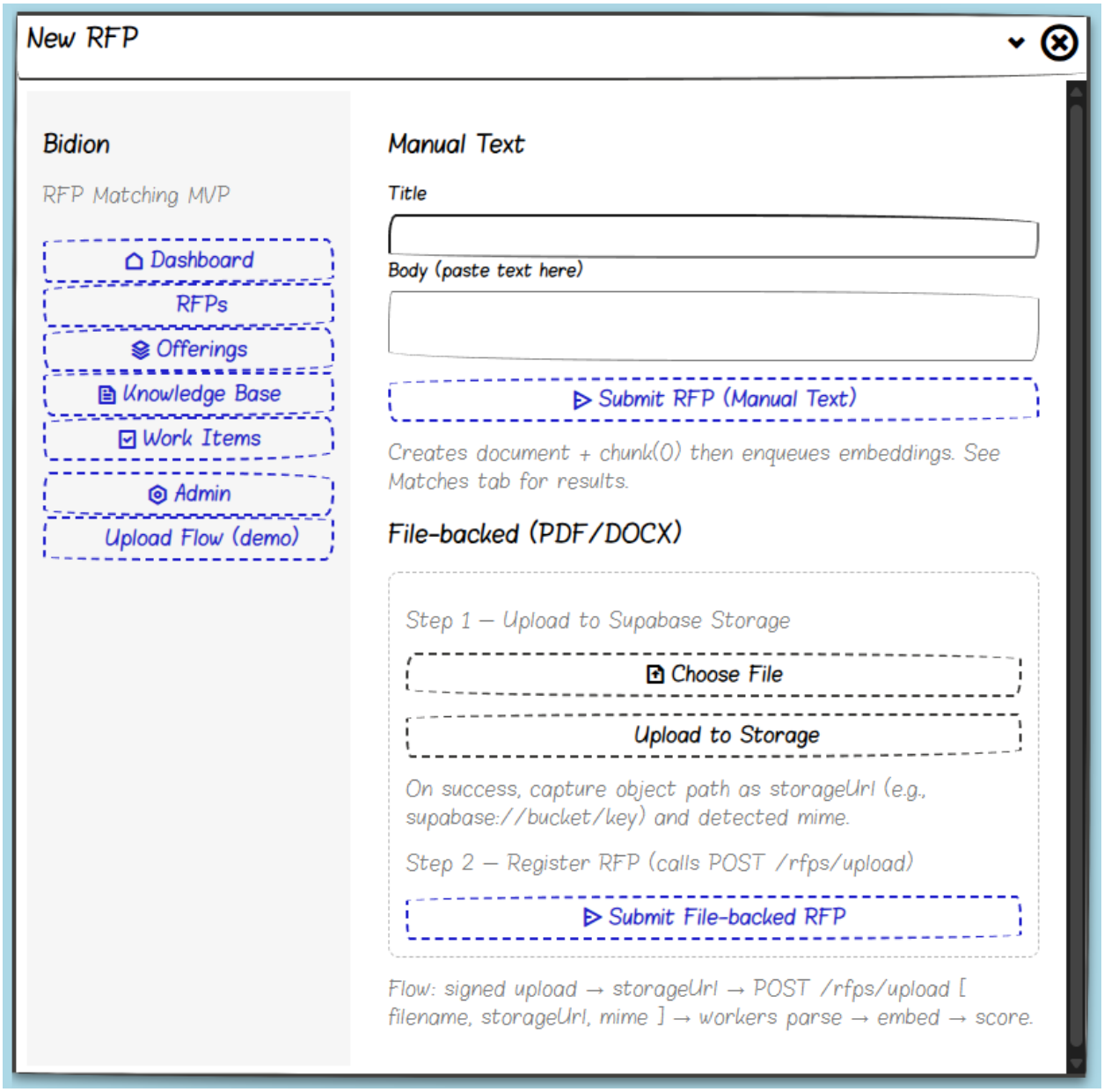


Figure 7: RFP creation screen supports two paths: (1) manual text (`POST /rfps` with `{title, bodyText}`), and (2) file-backed upload (signed upload to Supabase Storage then `POST /rfps/upload` with `{filename, storageUrl, mime}`). The server derives `orgId` from Supabase Auth in both cases; workers continue with *parse* → *embed* → *score*.

Listing 57: New RFP page: manual text and file-backed flows (aligned with API/auth)

```
'use client';
import { useState } from 'react';

export default function NewRfpPage() {
  // Manual text state
  const [title, setTitle] = useState('');
  const [body, setBody]   = useState('');
  // File-backed state
  const [file, setFile]   = useState(null);
  const [mime, setMime]   = useState('');
  const [objectPath, setObjectPath] = useState(null);
  const [rfpDocId, setRfpDocId] = useState();

  async function getAccessToken() {
    // In production, prefer a Next.js server route that sets an HTTP-only cookie.
    return localStorage.getItem('accessToken') || '';
  }

  // -------- Manual text -> POST /rfps { title, bodyText } --------
  async function submitManual() {
    const token = await getAccessToken();
    const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + '/rfps', {
      method: 'POST',
      headers: {
        'Content-Type':'application/json',
        'Authorization': token ? `Bearer ${token}` : ''
      },
      body: JSON.stringify({ title, bodyText: body }) // orgId derived server-side
    });
    if (!r.ok) throw new Error(`Create failed: ${r.status}`);
    const json = await r.json();
    setRfpDocId(json.documentId);
    alert('RFP created from text. Embeddings and scoring are running.');
  }

  // -------- File-backed flow: sign -> PUT -> POST /rfps/upload --------
  async function chooseFile(e) {
    const f = e.target.files?.[0];
    if (f) {
      setFile(f);
      setMime(f.type || 'application/octet-stream');
    }
  }
```

```
44
45  async function uploadToStorage() {
46    if (!file) return alert('Choose a file first');
47    const token = await getAccessToken();
48
49    // Step 1: sign
50    const sign = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + '/uploads/sign', {
51      method: 'POST',
52      headers: {
53        'Content-Type':'application/json',
54        'Authorization': token ? `Bearer ${token}` : ''
55      },
56      body: JSON.stringify({ filename: file.name, mime })
57    });
58    if (!sign.ok) throw new Error(`Sign failed: ${sign.status}`);
59    const { signedUrl, objectPath: path } = await sign.json();
60
61    // Step 2: direct upload (PUT) to signedUrl
62    const put = await fetch(signedUrl, {
63      method: 'PUT',
64      headers: { 'Content-Type': mime },
65      body: file
66    });
67    if (!put.ok) throw new Error(`Upload failed: ${put.status}`);
68
69    setObjectPath(path);
70    alert('Uploaded to storage.');
71  }
72
73  async function submitFileBacked() {
74    if (!objectPath || !file) return alert('Upload the file first');
75    const token = await getAccessToken();
76
77    // Step 3: register file-backed RFP
78    const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + '/rfps/upload', {
79      method: 'POST',
80      headers: {
81        'Content-Type':'application/json',
82        'Authorization': token ? `Bearer ${token}` : ''
83      },
84      body: JSON.stringify({
85        filename: file.name,
86        storageUrl: objectPath,  // e.g., supabase://bucket/key
87        mime
```

```
      })
    });
    if (!r.ok) throw new Error(`Register failed: ${r.status}`);
    const json = await r.json();
    setRfpDocId(json.documentId);
    alert('RFP registered from file. Parsing, embeddings, and scoring are running.');
  }

  return (
    <div style={{maxWidth: 880, margin: '2rem auto'}}>
      <h1>New RFP</h1>

      {/* Manual text section */}
      <section style={{marginTop: 16}}>
        <h3>Manual Text</h3>
        <input
          value={title}
          onChange={e=>setTitle(e.target.value)}
          placeholder="Title"
          style={{width:'100%', padding:8}}
        />
        <textarea
          value={body}
          onChange={e=>setBody(e.target.value)}
          placeholder="Paste the RFP body..."
          rows={10}
          style={{width:'100%', marginTop:8}}
        />
        <button onClick={submitManual} style={{marginTop:12}}>
          Submit RFP (Manual Text)
        </button>
      </section>

      {/* File-backed section */}
      <section style={{marginTop: 32}}>
        <h3>File-backed (PDF/DOCX)</h3>
        <input type="file" onChange={chooseFile} />
        <div style={{marginTop:8}}>
          <button onClick={uploadToStorage}>Upload to Storage</button>
          <span style={{marginLeft:8, color:'#777'}}>
            {objectPath ? `Uploaded: ${objectPath}` : 'Awaiting upload...'}
          </span>
        </div>
        <button onClick={submitFileBacked} style={{marginTop:12}}>
```

```
          Submit File-backed RFP
        </button>
      </section>

      {/* Optional matches preview after creation */}
      {rfpDocId && <Matches rfpId={rfpDocId} />}
    </div>
  );
}

function Matches({ rfpId }) {
  const [items, setItems] = useState([]);
  const [nextCursor, setNext] = useState(null);

  async function getAccessToken() {
    return localStorage.getItem('accessToken') || '';
  }

  async function refresh(cursor) {
    const token = await getAccessToken();
    const qs = cursor ? `?cursor=${encodeURIComponent(cursor)}` : '';
    const url = process.env.NEXT_PUBLIC_API_BASE_URL + `/rfps/${rfpId}/matches${qs}`;
    const r = await fetch(url, {
      headers: { 'Authorization': token ? `Bearer ${token}` : '' }
    });
    if (!r.ok) throw new Error(`Fetch matches failed: ${r.status}`);
    const json = await r.json();
    setItems(json.items || []);
    setNext(json.nextCursor || null);
  }

  return (
    <div style={{marginTop:24}}>
      <button onClick={()=>refresh(null)}>Refresh Matches</button>
      <ul>
        {items.map(x=>(
          <li key={x.offering_id}>
            <strong>{x.title}</strong> -- score {Number(x.score).toFixed(3)}
          </li>
        ))}
      </ul>
      {nextCursor && (
        <button onClick={()=>refresh(nextCursor)} style={{marginTop:8}}>
          Load more
```

```
        </button>
      )}
    </div>
  );
}
```

Remarks (auth, storage, and upload flow).

- **Supabase Auth**: in production obtain the access token server-side (Next.js route) and set HTTP-only cookies; the client-side bearer here is for clarity.
- **Do not send `orgId`**: the server derives `orgId` from the authenticated request and enforces RLS.
- **File-backed flow**: `POST /uploads/sign` returns a short-lived `signedUrl` and `objectPath`; the UI `PUT`s bytes to `signedUrl` and then calls `POST /rfps/upload` with `filename`, `storageUrl=objectPath`, and `mime`.

13.5 RFP Detail Page (`apps/web/app/rfps/[rfpId]/page.tsx`)

Integration map (wireframe ↔ API).

- **Overview tab** displays document metadata from `documents` (server-side load; org scoped by auth/RLS).
- **Matches tab** lists ranked offerings via `GET /rfps/:rfpId/matches` (see Section 11.2.6); supports cursor pagination.
- **Re-score now** triggers `POST /rfps/:rfpId/match` (see Section 11.2.7); admin/operator only; org derived from session.
- **Documents tab** *Upload* opens the upload flow (Supabase Storage) then calls `POST /rfps/upload` (see Section 11.2.2); worker parses → embeds → scores.
- **Assistant tab** calls the assistant draft API (not shown here) with context tags; server enforces RLS via session-derived `orgId`.
- **Auth headers**: Supabase Auth session → server issues/reads HTTP-only cookies (preferred) or bearer in dev.

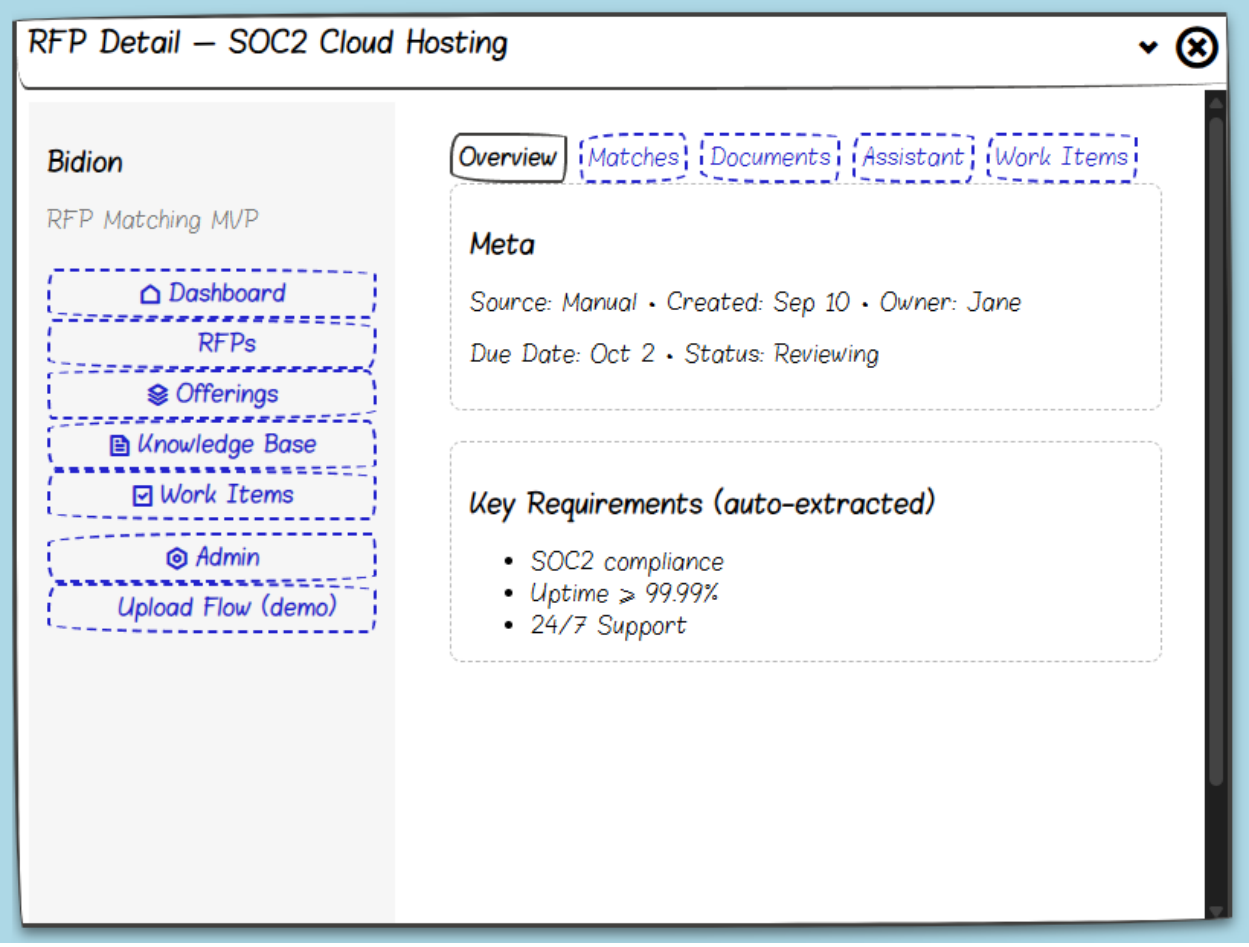


Figure 8: RFP Detail (tabs: Overview, Matches, Documents, Assistant, Work Items).

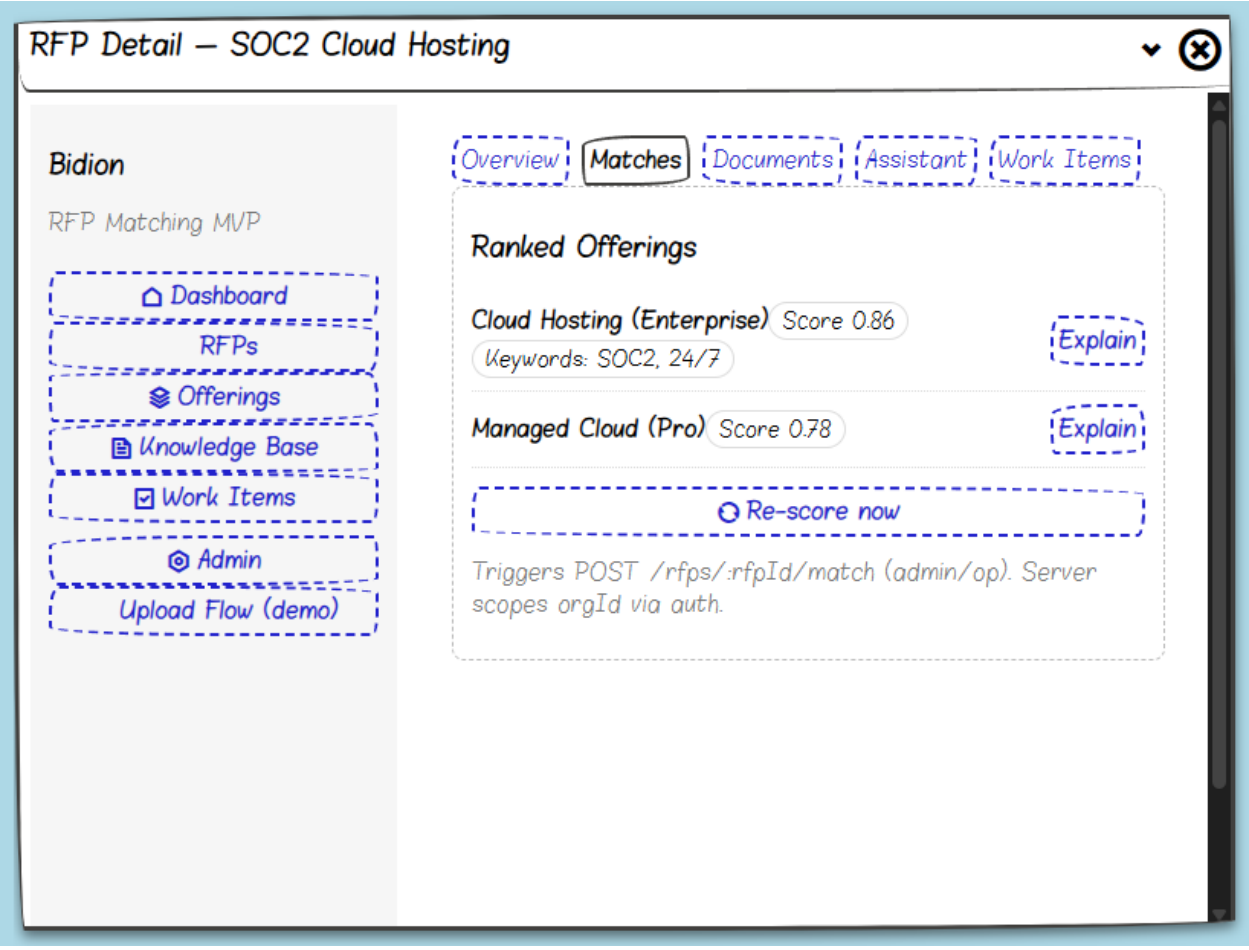


Figure 9: RFP Detail (tabs: Overview, Matches, Documents, Assistant, Work Items).

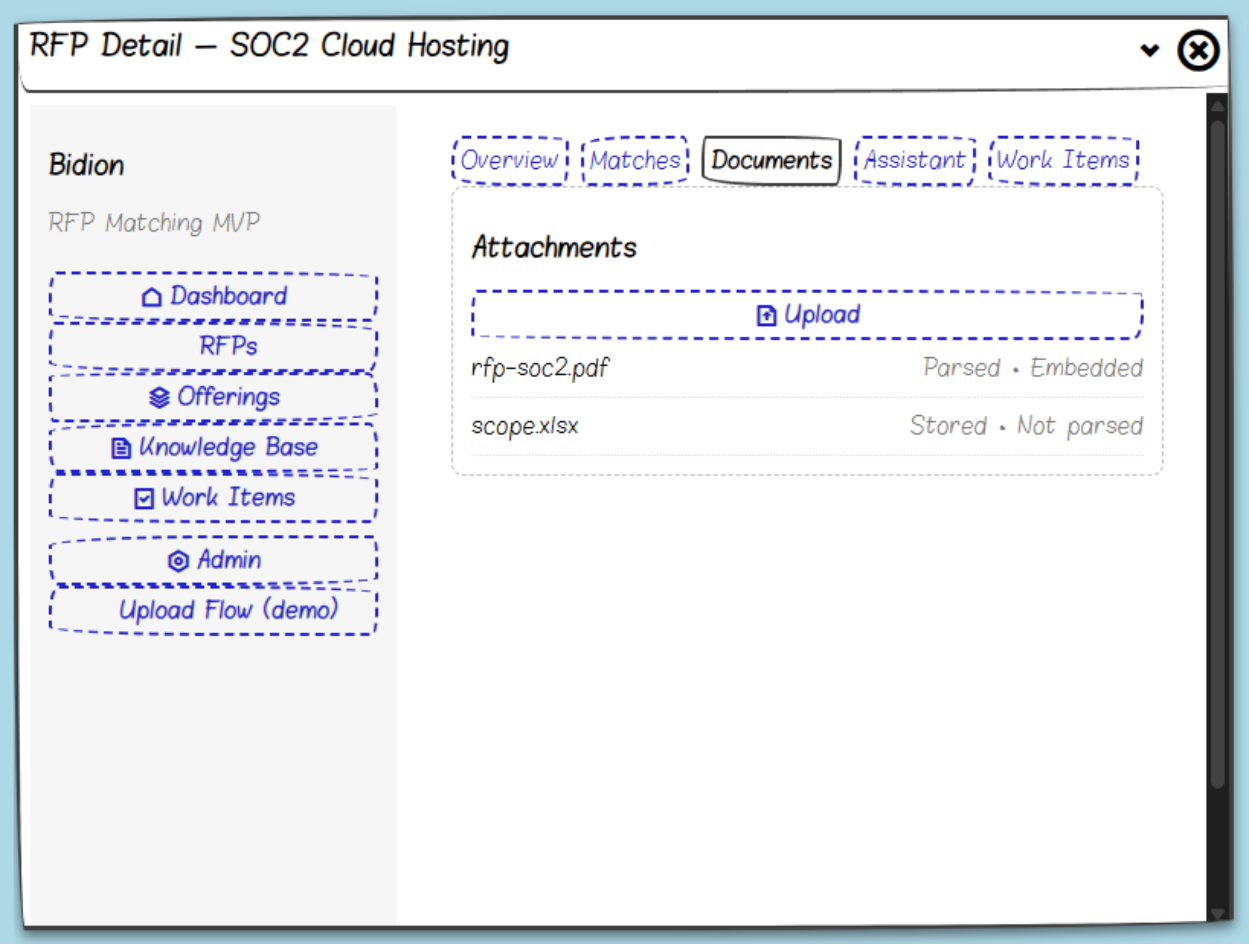


Figure 10: RFP Detail (tabs: Overview, Matches, Documents, Assistant, Work Items).

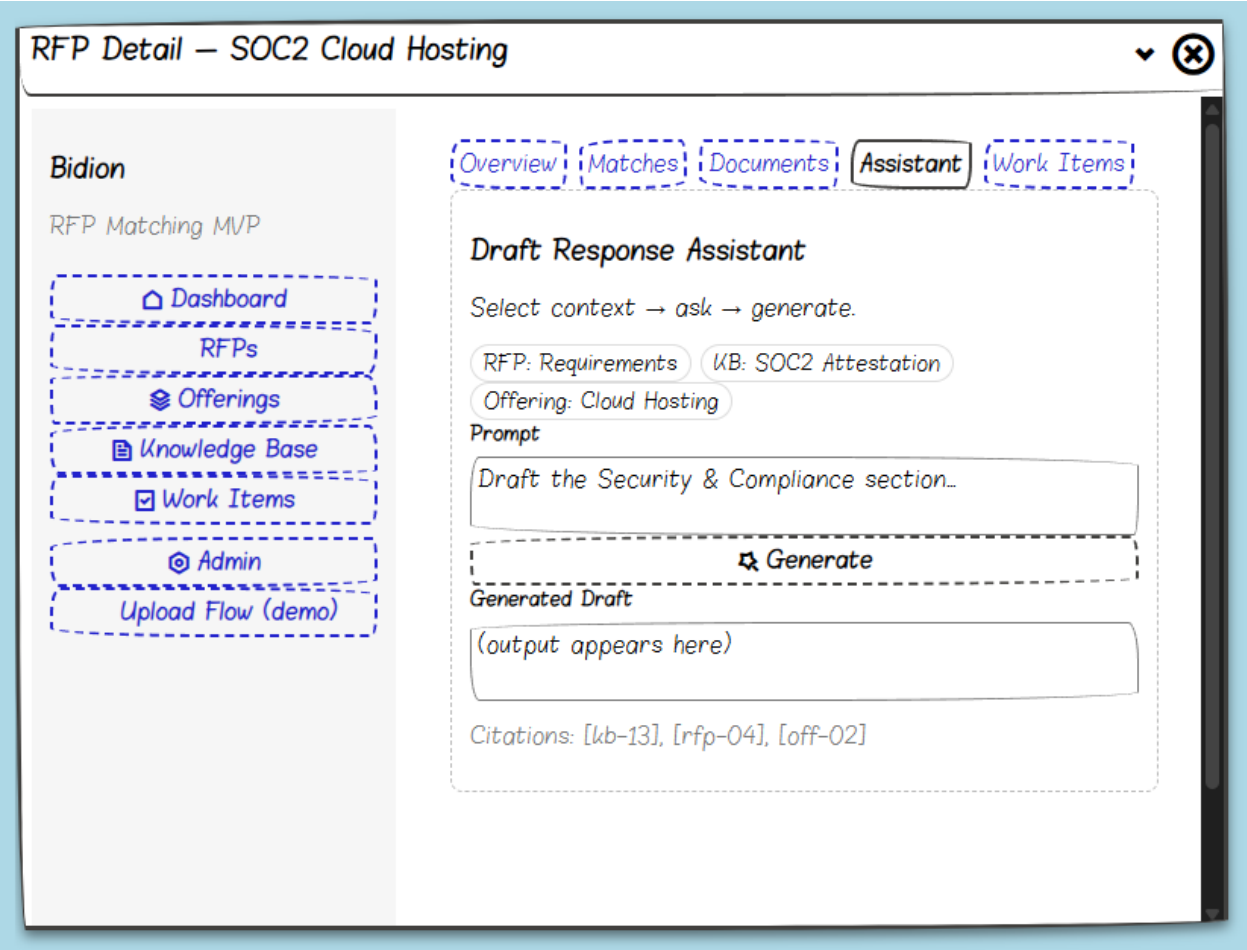


Figure 11: RFP Detail (tabs: Overview, Matches, Documents, Assistant, Work Items).

Figure 12: RFP Detail (tabs: Overview, Matches, Documents, Assistant, Work Items).

Listing 58: RFP Detail (Matches + Re-score actions aligned with API/auth)

```
'use client';
import { useEffect, useState } from 'react';
import { useParams } from 'next/navigation';

export default function RfpDetailPage() {
  const params = useParams();
  const rfpId = String(params.rfpId);
  const [meta, setMeta] = useState(null);

  async function token() {
    // Prefer HTTP-only cookies via Next.js route. Fallback for MVP:
    return localStorage.getItem('accessToken') || '';
  }

  useEffect(() => {
    // Load minimal metadata (you can expose an internal API or SSR this)
    (async () => {
      const t = await token();
      const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + `/rfps?limit=1&q=${
    encodeURIComponent(rfpId)}`, {
        headers: { 'Authorization': t ? `Bearer ${t}` : '' }
      });
      // In practice, add a dedicated GET /rfps/:rfpId if needed for richer meta
      setMeta({ id: rfpId });
    })();
  }, [rfpId]);

  return (
    <div style={{ maxWidth: 960, margin: '2rem auto' }}>
      <h1>RFP Detail</h1>
      <p style={{ color:'#777' }}>Tabs: Overview | Matches | Documents | Assistant | Work
     Items</p>
      <MatchesTab rfpId={rfpId}/>
      <RescorePanel rfpId={rfpId}/>
    </div>
  );
}

function MatchesTab({ rfpId }) {
  const [items, setItems] = useState([]);
  const [nextCursor, setNext] = useState(null);
  const [busy, setBusy] = useState(false);

```

```
  async function token() { return localStorage.getItem('accessToken') || ''; }

  async function load(cursor) {
    setBusy(true);
    try {
      const t = await token();
      const qs = cursor ? `?cursor=${encodeURIComponent(cursor)}` : '';
      const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + `/rfps/${rfpId}/matches${qs}`, {
        headers: { 'Authorization': t ? `Bearer ${t}` : '' }
      });
      if (!r.ok) throw new Error(`matches ${r.status}`);
      const json = await r.json(); // { items, nextCursor }
      setItems(cursor ? [...items, ...(json.items||[])] : (json.items||[]));
      setNext(json.nextCursor || null);
    } finally { setBusy(false); }
  }

  useEffect(() => { load(null); /* initial */ }, [rfpId]);

  return (
    <div className="card" style={{ marginTop: 16, padding: 12 }}>
      <h3>Ranked Offerings</h3>
      <ul>
        {items.map(x => (
          <li key={x.offering_id}>
            <strong>{x.title}</strong> -- score {Number(x.score).toFixed(3)}
          </li>
        ))}
      </ul>
      {nextCursor && (
        <button disabled={busy} onClick={() => load(nextCursor)} style={{ marginTop: 8 }}>
          {busy ? 'Loading...' : 'Load more'}
        </button>
      )}
    </div>
  );
}

function RescorePanel({ rfpId }) {
  const [busy, setBusy] = useState(false);
  const [k, setK] = useState(3);
  const [topN, setTopN] = useState(10);
```

```
  const [msg, setMsg] = useState('');

  async function token() { return localStorage.getItem('accessToken') || ''; }

  async function rescore() {
    setBusy(true); setMsg('');
    try {
      const t = await token();
      const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + `/rfps/${rfpId}/match
    `, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': t ? `Bearer ${t}` : ''
        },
        body: JSON.stringify({ k, topN, reason: 'manual' })
      });
      if (!r.ok) throw new Error(`rescore ${r.status}`);
      setMsg('Re-score enqueued.');
    } catch (e) {
      setMsg('Failed to enqueue re-score.');
    } finally { setBusy(false); }
  }

  return (
    <div className="card" style={{ marginTop: 16, padding: 12 }}>
      <h3>Re-score now</h3>
      <div style={{ display:'flex', gap:8, alignItems:'center' }}>
        <label>Top-N
          <select value={topN} onChange={e=>setTopN(Number(e.target.value))}>
            <option>10</option><option>20</option><option>50</option>
          </select>
        </label>
        <label>k
          <select value={k} onChange={e=>setK(Number(e.target.value))}>
            <option>3</option><option>5</option><option>10</option>
          </select>
        </label>
        <button disabled={busy} onClick={rescore}>
          {busy ? 'Enqueuing…' : 'Confirm Re-score'}
        </button>
      </div>
      {msg && <p style={{ color:'#777', marginTop:8 }}>{msg}</p>}
      <p style={{ color:'#777' }}>Calls POST /rfps/:rfpId/match; org is derived from auth
```

```
      on the server.</p>
127     </div>
128   );
129 }
```

Notes (auth, RLS, and uploads).

- **Auth/RLS**: The backend derives `orgId` from the Supabase session; do not send `orgId` from the client. DB RLS policies (documents/embeddings/matches) enforce tenant scope.
- **Pagination**: The Matches tab honors `nextCursor` from `GET /rfps/:rfpId/matches` and appends results.
- **Re-score**: Admin-only action; use feature flags/role checks on the server and optionally hide the control in the UI for non-admins.
- **Documents**: The Upload button should route to the Upload flow (Supabase Storage) and then call `POST /rfps/upload`.

13.6 RFP Re-score Page (`apps/web/app/rfps/[rfpId]/rescore/page.tsx`)

Integration map (wireframe ↔ API).

- **Confirm Re-score** → `POST /rfps/:rfpId/match` with JSON `{ k, topN }`. Do *not* send `orgId`; server derives it from Auth and applies RLS. See Section 11.2.7.
- **Defaults/bounds**: UI offers only bounded values (e.g., $k \in \{3, 5, 10\}$, `topN` $\in \{10, 20, 50\}$) to match server validation and prevent abuse.
- **Admin-only**: page/action should be gated by role/claims; server still enforces authorization.

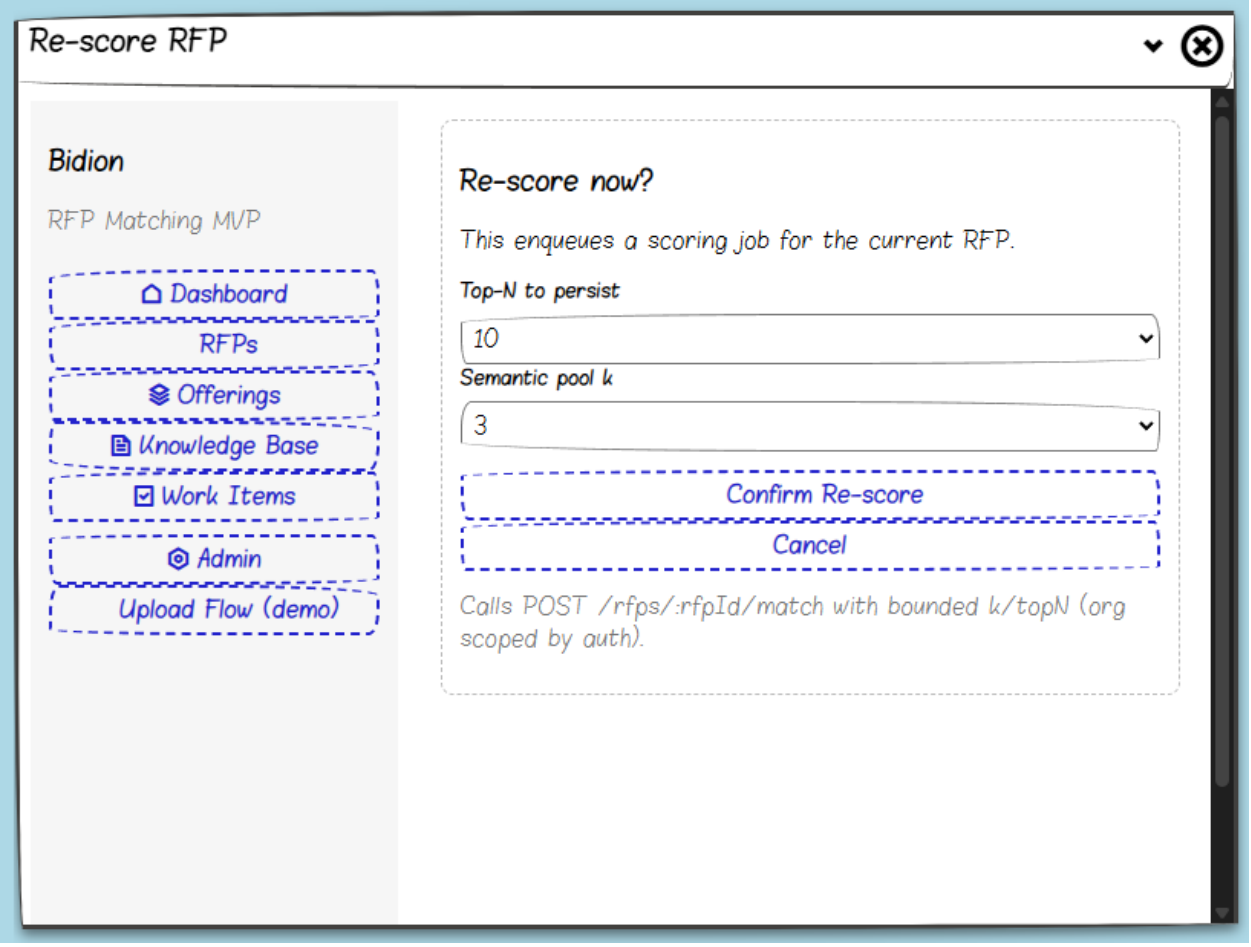


Figure 13: Re-score confirmation: user chooses bounded parameters (Top-N, semantic pool k) and confirms. Client calls `POST /rfps/:rfpId/match`. Server derives `orgId` from Supabase Auth and enqueues a scoring job (idempotent).

Listing 59: Rescore UI calling POST /rfps/:rfpId/match (aligned with API/auth)

```
'use client';
import { useState } from 'react';
import { useRouter, useParams } from 'next/navigation';

export default function RescorePage() {
  const router = useRouter();
  const params = useParams(); // expects { rfpId }
  const rfpId = String(params.rfpId || '');

  const [k, setK] = useState(3);
  const [topN, setTopN] = useState(10);
  const [busy, setBusy] = useState(false);
  const [err, setErr] = useState('');

  async function getAccessToken() {
    return localStorage.getItem('accessToken') || '';
  }

  async function confirmRescore() {
    setBusy(true); setErr('');
    try {
      const token = await getAccessToken();
      const r = await fetch(process.env.NEXT_PUBLIC_API_BASE_URL + `/rfps/${rfpId}/match
    `, {
        method: 'POST',
        headers: {
          'Content-Type':'application/json',
          'Authorization': token ? `Bearer ${token}` : ''
        },
        body: JSON.stringify({ k, topN }) // orgId derived server-side
      });
      if (!r.ok) {
        const t = await r.text();
        throw new Error(`Rescore failed: ${r.status} ${t}`);
      }
      router.push(`/rfps/${rfpId}`); // back to detail
    } catch (e) {
      setErr(e instanceof Error ? e.message : 'Unknown error');
    } finally {
      setBusy(false);
    }
  }

```

```
  return (
    <div style={{maxWidth:560, margin:'2rem auto'}}>
      <h1>Re-score RFP</h1>
      <p>This enqueues a scoring job for the current RFP.</p>

      <div style={{display:'grid', gap:8, marginTop:12, maxWidth:320}}>
        <label>
          <div>Top-N to persist</div>
          <select value={topN} onChange={e=>setTopN(Number(e.target.value))}>
            <option value={10}>10</option>
            <option value={20}>20</option>
            <option value={50}>50</option>
          </select>
        </label>
        <label>
          <div>Semantic pool k</div>
          <select value={k} onChange={e=>setK(Number(e.target.value))}>
            <option value={3}>3</option>
            <option value={5}>5</option>
            <option value={10}>10</option>
          </select>
        </label>
        <div style={{marginTop:12}}>
          <button onClick={confirmRescore} disabled={busy}>
            {busy ? 'Enqueuing...' : 'Confirm Re-score'}
          </button>
          <button onClick={()=>router.push(`/rfps/${rfpId}`)} style={{marginLeft:8}}>
            Cancel
          </button>
        </div>
        {err && <p style={{color:'crimson'}}>{err}</p>}
      </div>
    </div>
  );
}
```

Remarks (auth, idempotency, safety).

- **Auth & RLS**: server derives `orgId` from Supabase Auth and checks the RFP belongs to the org before enqueue.
- **Idempotent job**: server uses a stable `jobId` per (org, rfp) to avoid duplicate in-flight rescoring.
- **Readiness**: server may return `409 embeddings_not_ready` if embeddings are not yet persisted.

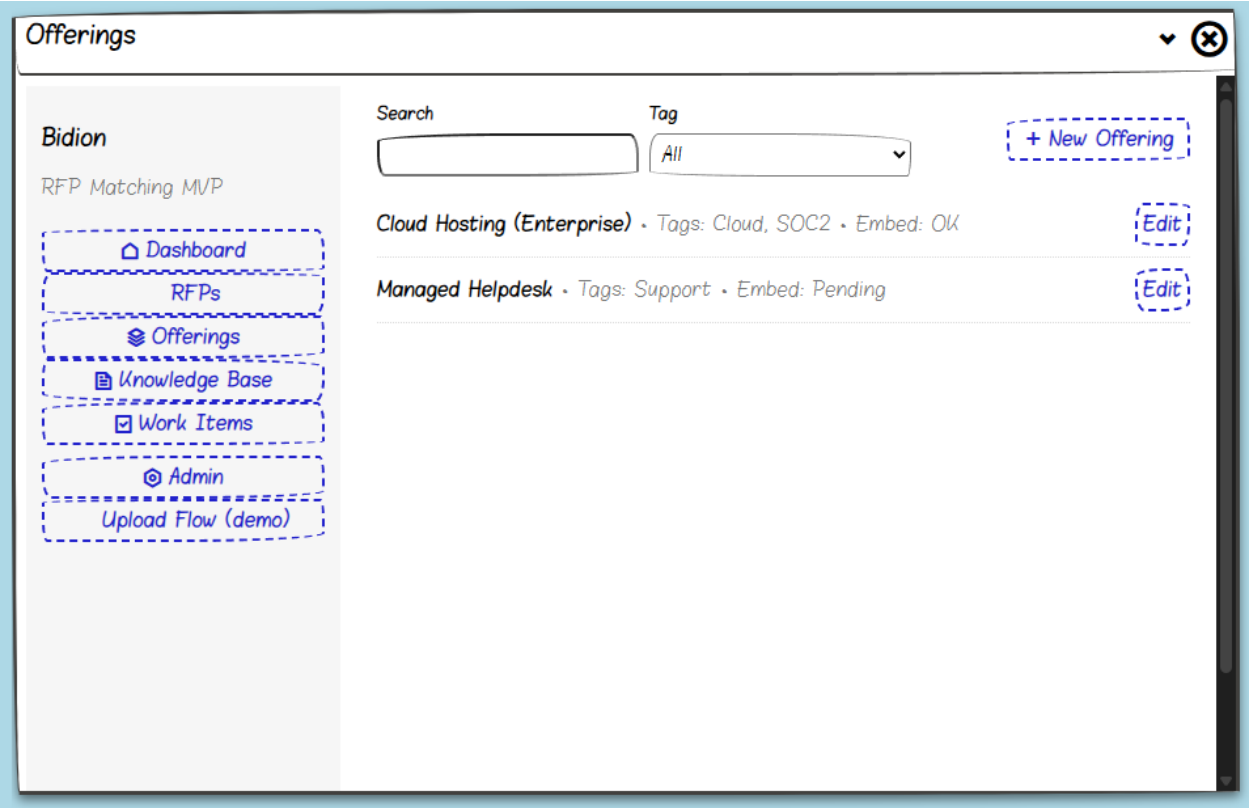


Figure 14: Offerings list (search, tag filter, embed status) with a "New Offering" CTA and per-row "Edit".

13.7 Offerings Page (`apps/web/app/offerings/page.tsx`)

Integration map (wireframe ↔ API).

- **List Offerings** → `GET /offerings?limit&cursor&q&tag`. Returns `{ items, nextCursor }` with fields `id`, `title`, `tags[]`, and `embedStatus`. Server derives `orgId` from Supabase Auth.
- **New Offering (manual text)** → `POST /offerings` with `{ title, description, tags[] }`; worker enqueues *embed*.
- **New Offering (file-backed)** → signed upload to Storage, then `POST /offerings/upload` with `{ title, storageUrl, mime, tags[] }`; worker runs *parse* → *embed*.
- **Auth headers**: use Supabase Auth bearer (`Authorization: Bearer <token>`) or HTTP-only cookies via Next.js server routes; never send `orgId` from the client.

Listing 60: Offerings list: search, tag filter, embed status (aligned with API/auth)

```
'use client';
import { useEffect, useState } from 'react';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

function useToken() {
```

```
  return typeof window !== 'undefined'
    ? (localStorage.getItem('accessToken') || '')
    : '';
}

export default function OfferingsPage() {
  const [q, setQ] = useState('');
  const [tag, setTag] = useState('All');
  const [items, setItems] = useState([]);
  const [nextCursor, setNext] = useState(null);
  const token = useToken();

  async function load(cursor) {
    const params = new URLSearchParams();
    if (q) params.set('q', q);
    if (tag && tag !== 'All') params.set('tag', tag);
    if (cursor) params.set('cursor', cursor);

    const r = await fetch(`${API}/offerings?${params.toString()}`, {
      headers: { Authorization: token ? `Bearer ${token}` : '' }
    });
    if (!r.ok) throw new Error(`List failed: ${r.status}`);
    const json = await r.json();
    setItems(cursor ? [...items, ...json.items] : json.items);
    setNext(json.nextCursor || null);
  }

  useEffect(() => { load(null); /* initial */ }, []); // eslint-disable-line

  return (
    <div style={{maxWidth: 960, margin: '2rem auto'}}>
      <h1>Offerings</h1>

      <div style={{display:'grid', gridTemplateColumns:'1fr 220px 120px', gap:8}}>
        <input placeholder="Search" value={q} onChange={e=>setQ(e.target.value)} />
        <select value={tag} onChange={e=>setTag(e.target.value)}>
          {['All','Cloud','Security','Support'].map(t => <option key={t}>{t}</option>)}
        </select>
        <button onClick={() => load(null)}>Apply</button>
      </div>

      <div style={{marginTop:16}}>
        {items.map(x => (
          <div key={x.id} style={{display:'flex', justifyContent:'space-between',
```

```
    borderBottom:'1px dotted #ddd', padding:'8px 0'}}>
51            <span>
52              <strong>{x.title}</strong>
53              <span style={{color:'#777'}}> > Tags: {x.tags?.join(', ') || '--'} > Embed:
     {x.embedStatus || '--'}</span>
54            </span>
55            <a href={`/offerings/${x.id}`}>Edit</a>
56          </div>
57        ))}
58      </div>
59
60      {nextCursor && (
61        <button onClick={() => load(nextCursor)} style={{marginTop:12}}>Load more</button
    >
62      )}
63
64      <div style={{marginTop:16}}>
65        <a href="/offerings/new">New Offering</a>
66      </div>
67    </div>
68  );
69 }
```

Remarks.

- **embedStatus**: compute via a LEFT JOIN from `documents(kind='offering')` to latest `embeddings` or a lightweight meta view; values like `Pending/OK/Failed`.
- **Pagination**: `cursor` is an opaque tuple (e.g., `created_at|id`) to ensure stable paging.

13.8 New Offering — Manual Text (`apps/web/app/offerings/new/page.tsx`)

Integration map (wireframe ↔ API).

- **Create Offering (manual text)** → `POST /offerings` with `{ title, description, tags[] }` (see §11.3.1). The server derives `orgId` from auth and enqueues *embed-chunks* for the materialized `document(kind='offering')`.
- **Navigation after create** → Return to list (`/offerings`) or open the new record.
- **Auth headers**: include `Authorization: Bearer <access_token>`, or rely on HTTP-only cookies if proxied through Next.js routes.

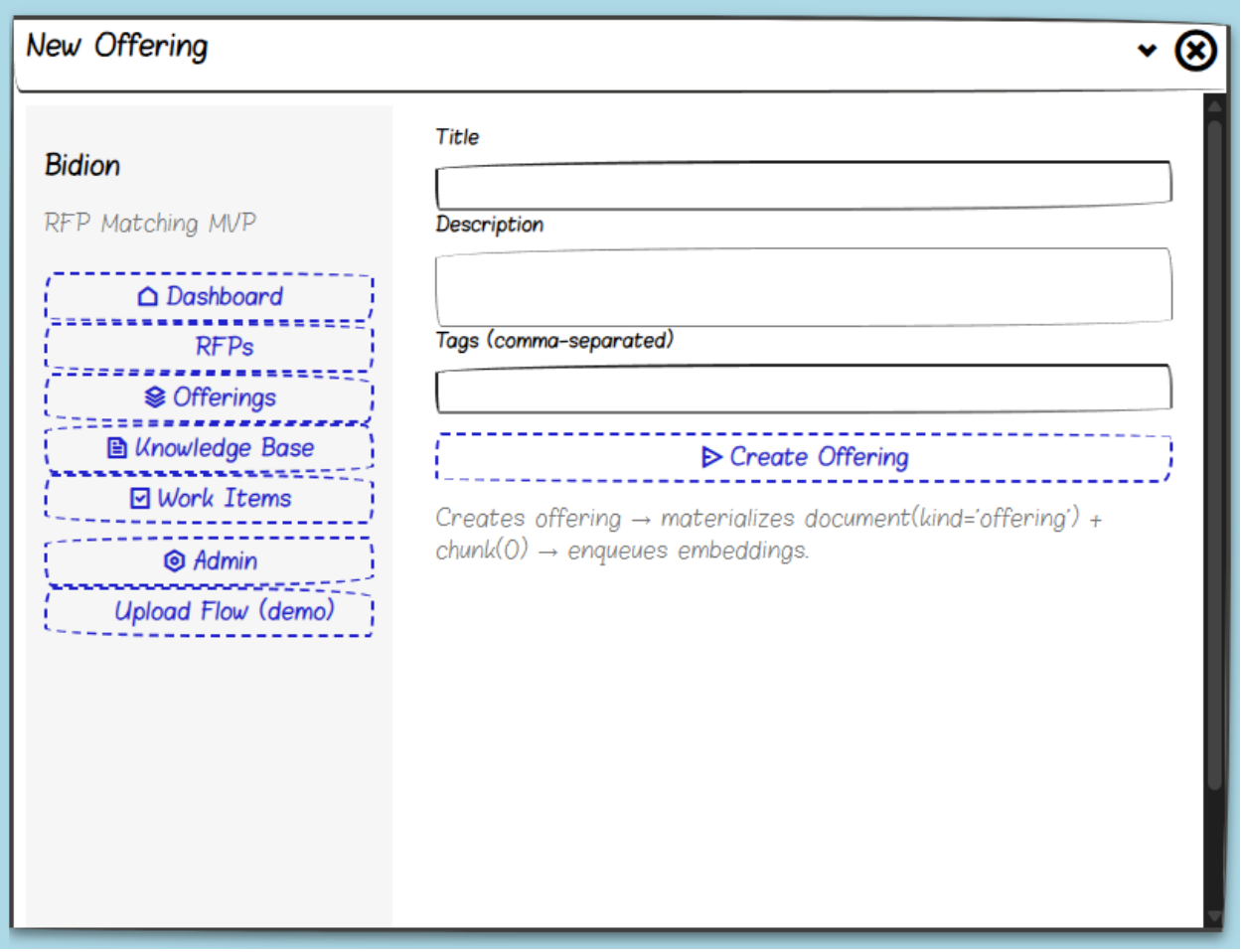


Figure 15: Create an offering from manual text.

Listing 61: Create a manual-text Offering (aligned with API/auth)

```
'use client';
import { useState } from 'react';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

export default function NewOfferingPage() {
  const [title, setTitle] = useState('');
  const [description, setDesc] = useState('');
  const [tags, setTags] = useState(''); // comma-separated

  async function token() {
    return localStorage.getItem('accessToken') || '';
  }

  async function createOffering() {
    const r = await fetch(`${API}/offerings`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': (await token()) ? `Bearer ${await token()}` : ''
      },
      body: JSON.stringify({
        title,
        description,
        tags: tags.split(',').map(t => t.trim()).filter(Boolean)
      })
    });
    if (!r.ok) throw new Error(`Create failed: ${r.status}`);
    // Optionally: const json = await r.json();
    // router.push('/offerings');
    alert('Offering created. Embeddings are being generated.');
  }

  return (
    <div style={{maxWidth:720, margin:'2rem auto'}}>
      <h1>New Offering</h1>
      <label>Title</label>
      <input value={title} onChange={e=>setTitle(e.target.value)} />
      <label>Description</label>
      <textarea rows={8} value={description} onChange={e=>setDesc(e.target.value)} />
      <label>Tags (comma-separated)</label>
      <input value={tags} onChange={e=>setTags(e.target.value)} />
      <button onClick={createOffering} style={{marginTop:12}}>
```

```
        Create Offering
      </button>
    </div>
  );
}
```

Remarks.

- **Do not send `orgId`**: server derives it from Supabase Auth; RLS protects multi-tenancy.
- **Embeddings**: the API materializes `document(kind='offering')` + `chunk(0)` and enqueues *embed*; no parsing step is needed for manual text.

13.9 New Offering — File-Backed (`apps/web/app/offerings/upload/page.tsx`)

Integration map (wireframe ↔ API).

- **Signed upload** → `POST /uploads/sign` with `{ filename, mime }` ⇒ returns `signedUrl`, `objectPath`.
- **Direct upload** → `PUT` bytes to `signedUrl`.
- **Register Offering (file-backed)** → `POST /offerings/upload` with `{ title, storageUrl=objectPath, mime, tags[] }` (see §11.3.2); server derives `orgId` and enqueues *parse-doc* → *embed*.
- **Auth headers**: include `Authorization: Bearer <access_token>` (or use HTTP-only cookies via Next.js routes).

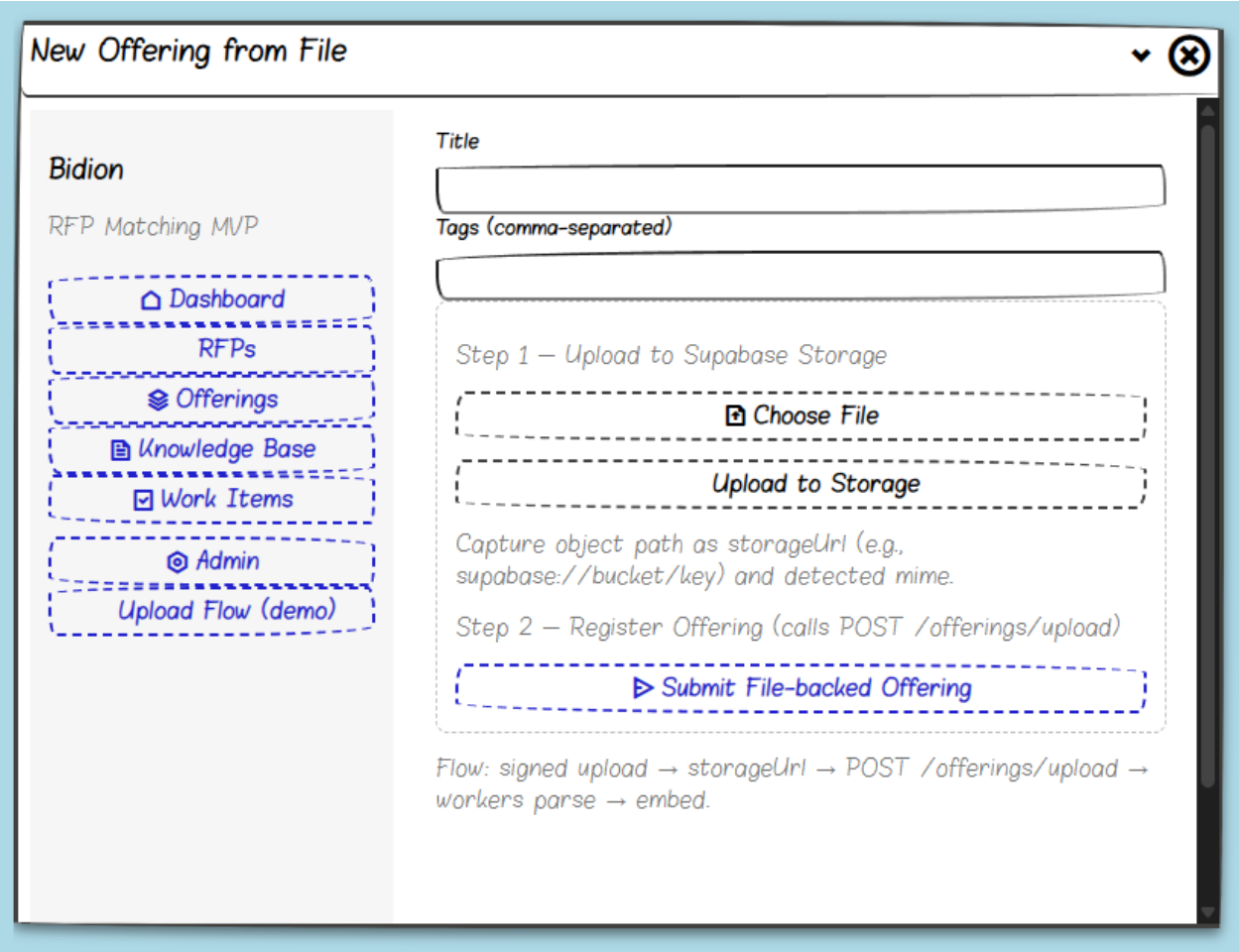


Figure 16: Upload to Supabase Storage, then register via POST /offerings/upload title, storageUrl, mime, tags[] — server derives orgId.

Listing 62: Create a file-backed Offering (signed upload + register)

```
'use client';
import { useState } from 'react';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

export default function NewOfferingUploadPage() {
  const [title, setTitle] = useState('');
  const [tags, setTags] = useState('');      // comma-separated
  const [file, setFile] = useState(null);
  const [mime, setMime] = useState('');
  const [objectPath, setObjectPath] = useState(null);

  async function token() {
    return localStorage.getItem('accessToken') || '';
  }

  function onChoose(e) {
    const f = e.target.files?.[0];
    if (f) {
      setFile(f);
      setMime(f.type || 'application/octet-stream');
    }
  }

  async function uploadToStorage() {
    if (!file) return alert('Choose a file first');
    // 1) sign
    const sign = await fetch(`${API}/uploads/sign`, {
      method: 'POST',
      headers: {
        'Content-Type':'application/json',
        'Authorization': (await token()) ? `Bearer ${await token()}` : ''
      },
      body: JSON.stringify({ filename: file.name, mime })
    });
    if (!sign.ok) throw new Error(`Sign failed: ${sign.status}`);
    const { signedUrl, objectPath: path } = await sign.json();
    // 2) PUT to signedUrl
    const put = await fetch(signedUrl, {
      method: 'PUT',
      headers: { 'Content-Type': mime },
      body: file
    });
```

```
    if (!put.ok) throw new Error(`Upload failed: ${put.status}`);
    setObjectPath(path);
    alert('Uploaded to storage.');
  }

  async function registerOffering() {
    if (!objectPath) return alert('Upload the file first');
    const r = await fetch(`${API}/offerings/upload`, {
      method: 'POST',
      headers: {
        'Content-Type':'application/json',
        'Authorization': (await token()) ? `Bearer ${await token()}` : ''
      },
      body: JSON.stringify({
        title,
        storageUrl: objectPath,  // e.g., supabase://bucket/key
        mime,
        tags: tags.split(',').map(t => t.trim()).filter(Boolean)
      })
    });
    if (!r.ok) throw new Error(`Register failed: ${r.status}`);
    // Optionally: const json = await r.json();
    // router.push('/offerings');
    alert('Offering registered. Parsing and embeddings are running.');
  }

  return (
    <div style={{maxWidth:720, margin:'2rem auto'}}>
      <h1>New Offering from File</h1>
      <label>Title</label>
      <input value={title} onChange={e=>setTitle(e.target.value)} />
      <label>Tags (comma-separated)</label>
      <input value={tags} onChange={e=>setTags(e.target.value)} />
      <div style={{marginTop:12}}>
        <input type="file" onChange={onChoose} />
      </div>
      <div style={{marginTop:8}}>
        <button onClick={uploadToStorage}>Upload to Storage</button>
        <span style={{marginLeft:8, color:'#777'}}>
          {objectPath ? `Uploaded: ${objectPath}` : 'Awaiting upload...'}
        </span>
      </div>
      <button onClick={registerOffering} style={{marginTop:12}}>
        Submit File-backed Offering
```

```
      </button>
    </div>
  );
}
```

Remarks.

- **Do not send `orgId`** from the client. The backend derives it from auth and enforces RLS.
- **Worker chain**: file-backed flow uses *parse-doc* → *embed-chunks*. Manual text skips parsing and goes straight to *embed-chunks*.
- **Signed upload**: `/uploads/sign` returns a short-lived `signedUrl` and the canonical `objectPath` to persist as `storageUrl`.

13.10 Edit Offering Page (`apps/web/app/offerings/[id]/page.tsx`)

Integration map (wireframe ↔ API).

- **Load Offering** → `GET /offerings/:id` to populate `title`, `description`, `tags[]`.
- **Save Changes** → `PATCH /offerings/:id` with partial body. If text changes, server re-materializes the offering's `document/chunk(0)` and re-enqueues *embed*.
- **Re-embed Now** → `POST /offerings/:id/reembed` (no body). Forces an embed job for the linked `document(kind='offering')` even if no text changed.

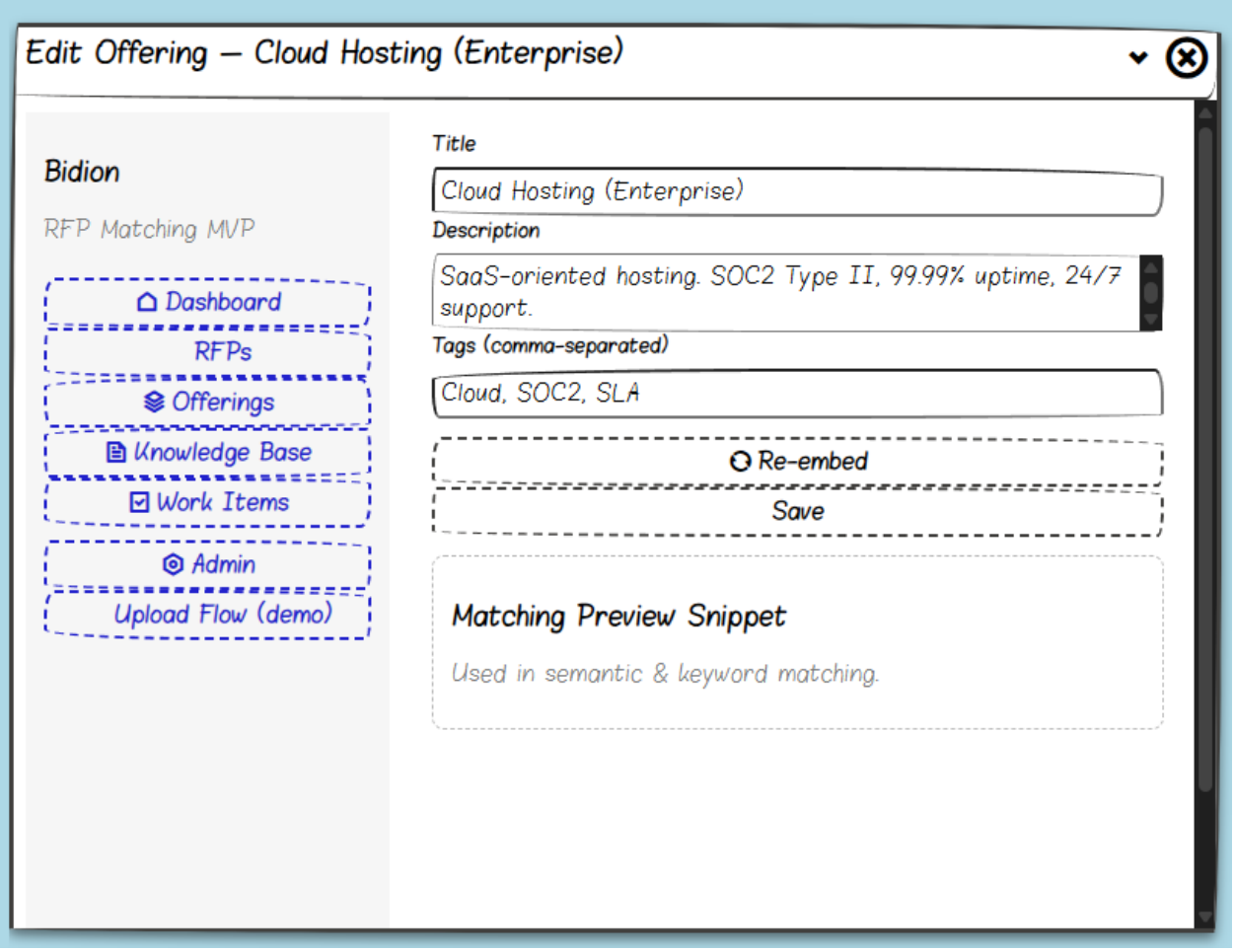


Figure 17: Edit Offering page (title, description, tags) with *Save* and *Re-embed* actions. Saving calls `PATCH /offerings/:id`; when text changes, the server refreshes `document/chunk(0)` and re-enqueues embeddings, with a preview snippet shown for matching context.

Listing 63: Offering edit — load, save (PATCH), and force re-embed (POST /reembed)

```
'use client';
import { useEffect, useState } from 'react';
import { useParams, useRouter } from 'next/navigation';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

export default function OfferingEditPage() {
  const { id } = useParams();
  const router = useRouter();
  const [title, setTitle] = useState('');
  const [description, setDesc] = useState('');
  const [tags, setTags] = useState('');
  const [busy, setBusy] = useState(false);

  const token = typeof window !== 'undefined'
    ? (localStorage.getItem('accessToken') || '')
    : '';

  useEffect(() => {
    (async () => {
      const r = await fetch(`${API}/offerings/${id}`, {
        headers: { Authorization: token ? `Bearer ${token}` : '' }
      });
      if (!r.ok) throw new Error(`Load failed: ${r.status}`);
      const x = await r.json();
      setTitle(x.title || '');
      setDesc(x.description || '');
      setTags((x.tags || []).join(', '));
    })();
  }, [id]); // eslint-disable-line

  async function save() {
    setBusy(true);
    const r = await fetch(`${API}/offerings/${id}`, {
      method: 'PATCH',
      headers: {
        'Content-Type':'application/json',
        'Authorization': token ? `Bearer ${token}` : ''
      },
      body: JSON.stringify({
        title,
        description,
        tags: tags.split(',').map(t => t.trim()).filter(Boolean)
```

```
      })
    });
    setBusy(false);
    if (!r.ok) throw new Error(`Save failed: ${r.status}`);
    router.push('/offerings');
  }

  async function reembed() {
    setBusy(true);
    const r = await fetch(`${API}/offerings/${id}/reembed`, {
      method: 'POST',
      headers: { 'Authorization': token ? `Bearer ${token}` : '' }
    });
    setBusy(false);
    if (!r.ok) throw new Error(`Re-embed failed: ${r.status}`);
    alert('Re-embed job enqueued.');
  }

  return (
    <div style={{maxWidth: 720, margin: '2rem auto'}}>
      <h1>Edit Offering</h1>
      <label>Title</label>
      <input value={title} onChange={e=>setTitle(e.target.value)} />
      <label>Description</label>
      <textarea rows={8} value={description} onChange={e=>setDesc(e.target.value)} />
      <label>Tags (comma-separated)</label>
      <input value={tags} onChange={e=>setTags(e.target.value)} />
      <div style={{display:'flex', gap:8, marginTop:12}}>
        <button onClick={reembed} disabled={busy}>Re-embed</button>
        <button onClick={save} disabled={busy}>{busy ? 'Saving...' : 'Save'}</button>
      </div>
      <p style={{color:'#777', marginTop:8}}>
        Save → PATCH /offerings/:id (text changes auto re-embed). Re-embed → POST /
    offerings/:id/reembed.
      </p>
    </div>
  );
}
```

Remarks.

- **Auth & RLS**: backend derives `orgId` from Supabase Auth; do not accept client-sent `orgId`.
- **Re-embed policy**: `PATCH` auto-enqueues embed on text change; `POST /offerings/:id/reembed` provides an explicit control to refresh embeddings without editing content.

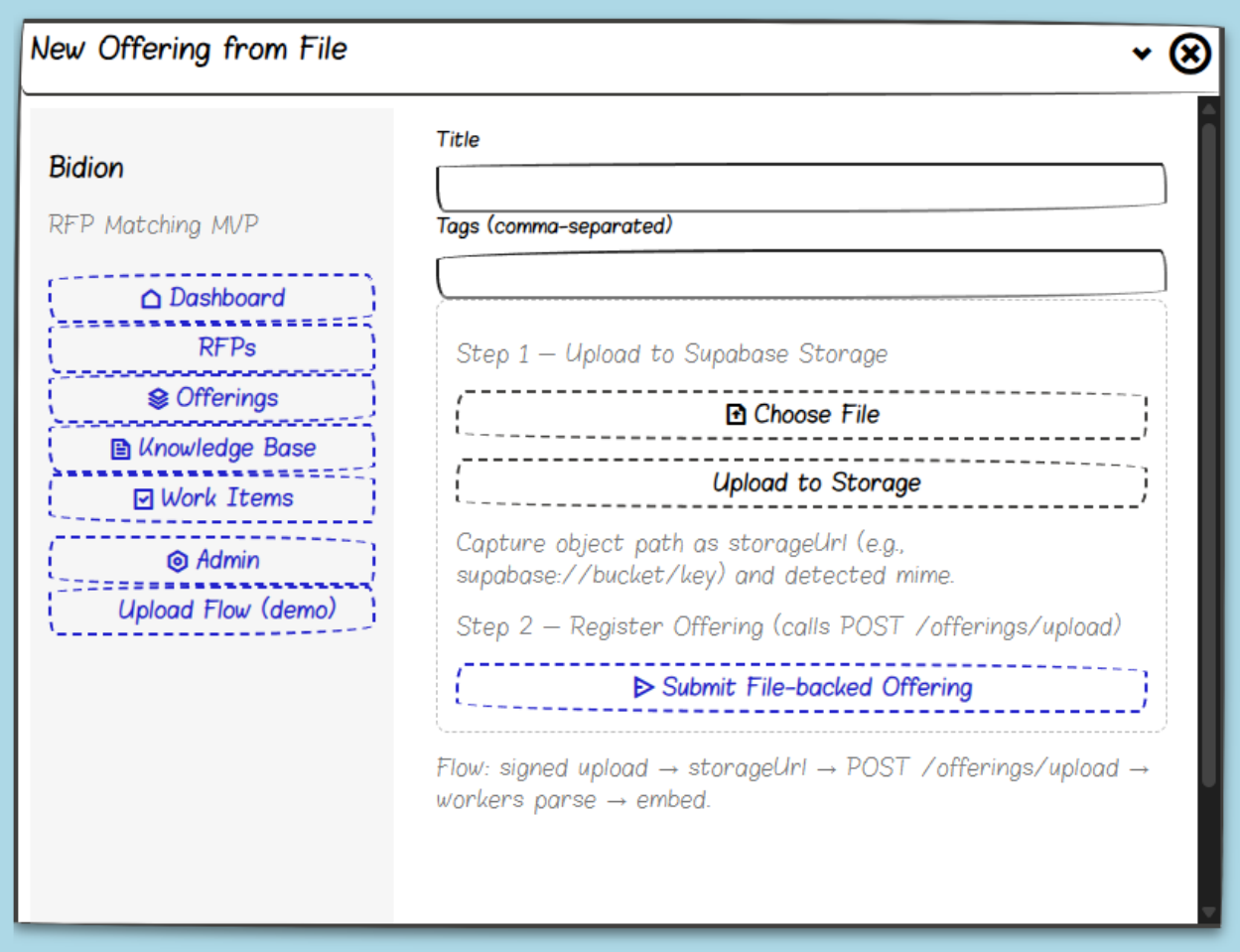


Figure 18: Knowledge Base list with upload action, search, kind/status filters, and per-row parse/embed status.

13.11 Knowledge Base Page (`apps/web/app/kb/page.tsx`)

Integration map (wireframe ↔ API).

- **List KB docs** → `GET /kb?q&kind&status&limit&offset`. Returns `{ items, limit, offset }`. Each item includes `filename`, `mime` (used to derive "Kind" = PDF/DOCX/...) and a computed `status` (e.g., Parsed/Embedded/Pending) based on worker progress. *Server derives `orgId` from auth (RLS).*
- **Upload (file-backed)**:
  1. `POST /uploads/sign` with `{ filename, mime }` → returns `signedUrl`, `objectPath`.
  2. Client `PUT` bytes to `signedUrl`.
  3. `POST /kb/upload` with `{ title, storageUrl=objectPath, mime }`; server enqueues *parse → embed. Do not send `orgId`; derived from auth.*
- **Filters**. UI "Kind" maps to `mime` families (e.g., PDF, DOCX); UI "Status" maps to server-computed pipeline status. Both are supported by `GET /kb` to match the wireframe controls.

- **Auth headers**. Use Supabase Auth bearer `Authorization: Bearer <token>` or rely on HTTP-only cookies via Next.js server routes.

Listing 64: KB list page with search, kind/status filters (aligned with GET /kb)

```
'use client';
import { useEffect, useState } from 'react';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

function useToken() {
  return typeof window !== 'undefined'
    ? (localStorage.getItem('accessToken') || '')
    : '';
}

export default function KnowledgeBasePage() {
  const token = useToken();
  const [q, setQ] = useState('');
  const [kind, setKind] = useState('All');     // All | PDF | DOCX
  const [status, setStatus] = useState('All'); // All | Parsed | Embedded | Pending

  const [items, setItems] = useState([]);
  const [limit, setLimit] = useState(20);
  const [offset, setOffset] = useState(0);

  async function load(nextOffset = 0) {
    const params = new URLSearchParams();
    if (q) params.set('q', q);
    if (kind && kind !== 'All') params.set('kind', kind);
    if (status && status !== 'All') params.set('status', status);
    params.set('limit', String(limit));
    params.set('offset', String(nextOffset));

    const r = await fetch(`${API}/kb?${params.toString()}`, {
      headers: { Authorization: token ? `Bearer ${token}` : '' }
    });
    if (!r.ok) throw new Error(`KB list failed: ${r.status}`);
    const json = await r.json();

    setItems(nextOffset === 0 ? json.items : [...items, ...json.items]);
    setOffset(nextOffset);
  }

  useEffect(() => { load(0); /* initial */ }, []); // eslint-disable-line
```

```
41
42    return (
43      <div style={{maxWidth: 960, margin: '2rem auto'}}>
44        <h1>Knowledge Base</h1>
45
46        <div style={{display:'grid', gridTemplateColumns:'1fr 160px 160px 160px', gap:8}}>
47          <input placeholder="Search" value={q} onChange={e=>setQ(e.target.value)} />
48          <select value={kind} onChange={e=>setKind(e.target.value)}>
49            {['All','PDF','DOCX'].map(k => <option key={k}>{k}</option>)}
50          </select>
51          <select value={status} onChange={e=>setStatus(e.target.value)}>
52            {['All','Parsed','Embedded','Pending'].map(s => <option key={s}>{s}</option>)}
53          </select>
54          <a href="/upload-flow">
55            <button>Upload Documents</button>
56          </a>
57        </div>
58
59        <div style={{marginTop:16}}>
60          {items.map(x => (
61            <div key={x.document_id} style={{display:'flex', justifyContent:'space-between'
      , borderBottom:'1px dotted #ddd', padding:'8px 0'}}>
62              <span>{x.filename}</span>
63              <span style={{color:'#777'}}>{x.status}</span>
64            </div>
65          ))}
66        </div>
67
68        <div style={{marginTop:12, display:'flex', gap:8}}>
69          <button onClick={() => load(0)}>Apply</button>
70          <button onClick={() => load(offset + limit)}>Load more</button>
71        </div>
72      </div>
73    );
74  }
```

Remarks (alignment with API design).

- **Endpoint coverage.** The UI uses `GET /kb` with `q`, `kind`, and `status` filters to match the wireframe controls; upload follows the signed-URL pattern then `POST /kb/upload`.
- **Kinds.** "Kind" in the UI is derived from `mime` (e.g., `application/pdf` → PDF). The server exposes a `kind` filter that maps to `mime` families.
- **Status.** "Parsed / Embedded / Pending" is computed server-side from worker progress (documents → chunks → embeddings). Exposing a `status` filter keeps the FE simple.

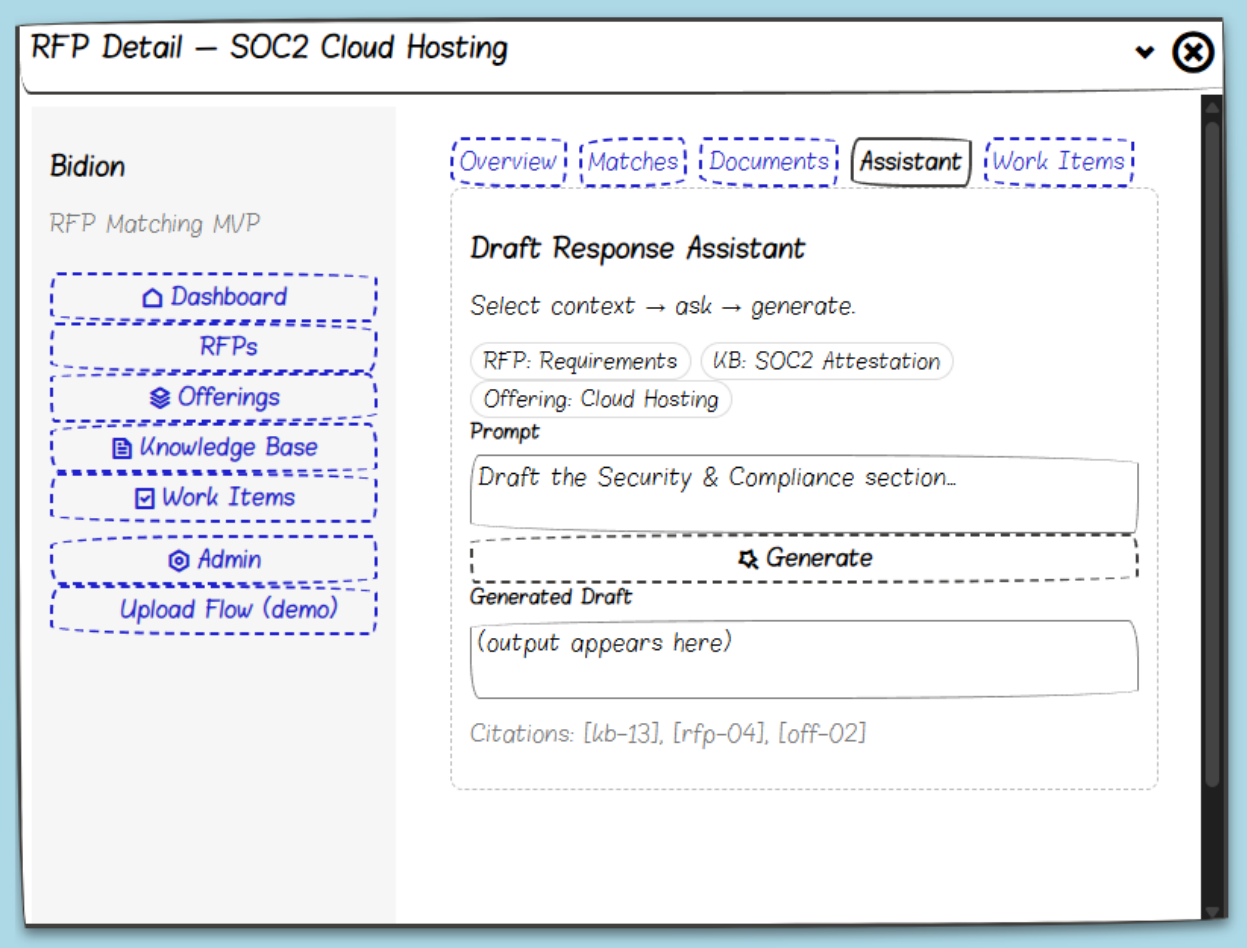


Figure 19: Assistant tab: select context, enter a prompt, generate a draft, and view citations.

- **Auth/RLS.** `orgId` is not accepted from the client; the server derives it from Supabase Auth and enforces RLS.

13.12 RFP Assistant Tab (`apps/web/app/rfps/[id]/assistant/page.tsx`)

Integration map (wireframe ↔ API).

- **Generate Draft** → `POST /assistant/draft` with `{ rfpId, question }` (see Section 11.5.1). The server derives `orgId` from auth and applies RLS.
- **Context chips (RFP / KB / Offering)** — visual only. Retrieval scope is determined server-side from `rfpId`; optional future flags can be added (e.g., `includeKb`, `includeOfferings`) if you want the UI to toggle sources.
- **Citations** — show IDs returned from `sources[]` (e.g., `[kb-13], [rfp-04], [off-02]`). These are hints; linking back to documents is optional.
- **Auth headers** — Supabase Auth bearer (`Authorization: Bearer <token>`) or HTTP-only cookies via Next.js server routes. Do not send `orgId` from the client.

Listing 65: Assistant tab UI calling POST /assistant/draft (aligned with API/auth)

```tsx
// apps/web/app/rfps/[id]/assistant/page.tsx
'use client';
import { useState } from 'react';
import { useParams } from 'next/navigation';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

export default function RfpAssistantPage() {
  const { id: rfpId } = useParams();
  const [question, setQuestion] = useState('Draft the Security & Compliance section...');
  const [draft, setDraft] = useState('');
  const [sources, setSources] = useState<string[]>([]);
  const [busy, setBusy] = useState(false);

  function token() {
    return typeof window !== 'undefined' ? (localStorage.getItem('accessToken') || '') :
    '';
  }

  async function generate() {
    if (!rfpId) return;
    setBusy(true);
    setDraft('');
    setSources([]);
    const r = await fetch(`${API}/assistant/draft`, {
      method: 'POST',
      headers: {
        'Content-Type':'application/json',
        'Authorization': token() ? `Bearer ${token()}` : ''
      },
      body: JSON.stringify({ rfpId, question }) // orgId derived server-side
    });
    setBusy(false);
    if (!r.ok) {
      const err = await r.json().catch(() => ({}));
      alert(`Generate failed: ${r.status} ${err?.error || ''}`);
      return;
    }
    const json = await r.json();
    setDraft(json.text || '');
    setSources(json.sources || []);
  }

```

```
  return (
    <div style={{maxWidth: 960, margin: '2rem auto'}}>
      <div style={{border:'1px dashed #bbb', borderRadius:8, padding:16, background:'#fff'}}>
        <h3>Draft Response Assistant</h3>
        <p style={{color:'#777'}}>Select context → ask → generate.</p>
        <div style={{marginBottom:12}}>
          <span style={pill}>RFP: Requirements</span>
          <span style={pill}>KB: SOC2 Attestation</span>
          <span style={pill}>Offering: Cloud Hosting</span>
        </div>

        <label style={{display:'block', fontWeight:600}}>Prompt</label>
        <textarea
          value={question}
          onChange={e=>setQuestion(e.target.value)}
          rows={6}
          placeholder="Ask a question or provide drafting instructions..."
          style={{width:'100%', marginTop:4}}
        />

        <button onClick={generate} disabled={busy} style={{marginTop:12}}>
          {busy ? 'Generating...' : 'Generate'}
        </button>

        <label style={{display:'block', fontWeight:600, marginTop:16}}>Generated Draft</label>
        <textarea
          value={draft}
          readOnly
          rows={12}
          placeholder="(output appears here)"
          style={{width:'100%', marginTop:4}}
        />

        {sources?.length > 0 && (
          <p style={{color:'#777', marginTop:8}}>
            Citations: {sources.map((s, i) => `[${s}]`).join(', ')}
          </p>
        )}
      </div>
    </div>
  );
}
```

```
85
86 const pill: React.CSSProperties = {
87   display:'inline-block',
88   padding:'2px 8px',
89   border:'1px solid #ddd',
90   borderRadius:999,
91   fontSize:12,
92   marginRight:6
93 };
```

Remarks.

- **Minimal client contract.** The UI only sends `rfpId` and `question`; retrieval joins RFP chunks with KB/Offering embeddings under the caller's org via RLS.
- **Citations.** Treat `sources[]` as lightweight references suitable for pill-style display (e.g., `[kb-13]`). You can later resolve these to deep-links.
- **Extensibility.** If you later add toggles for "Include KB / Include Offerings," extend the request to `{ rfpId, question, includeKb, includeOfferings }` and gate retrieval paths server-side.

13.13 Match Explain Page (`apps/web/app/rfps/[id]/matches/[offeringId]/explain/page.tsx`

Integration map (wireframe ↔ API).

- **Explain a specific match** → `GET /rfps/:rfpId/matches/:offeringId/explain`. Returns a structured breakdown:
  - `semantic.score` (0–1) and optional top contributing chunk IDs.
  - `keywords.delta` (signed contribution), with `hits[]` and per-hit terms.
  - `rules.delta` (signed contribution), with `requiredOK` / `forbiddenOK` and messages.
  - `final` (the persisted match score).
  - `highlights[]` — display-ready strings (e.g., "*SOC2 Type II* matched").
- **Back navigation** → client routes back to `/rfps/:rfpId` (no API).
- **Auth headers**: Supabase Auth bearer (`Authorization: Bearer <token>`); server derives `orgId` via RLS and validates that `rfpId` & `offeringId` belong to the caller's org.

Listing 66: Explain page (fetch & render breakdown + highlights)

```
1 // apps/web/app/rfps/[id]/matches/[offeringId]/explain/page.tsx
2 'use client';
3 import { useEffect, useState } from 'react';
4 import { useParams, useRouter } from 'next/navigation';
5
6 const API = process.env.NEXT_PUBLIC_API_BASE_URL;
7
```

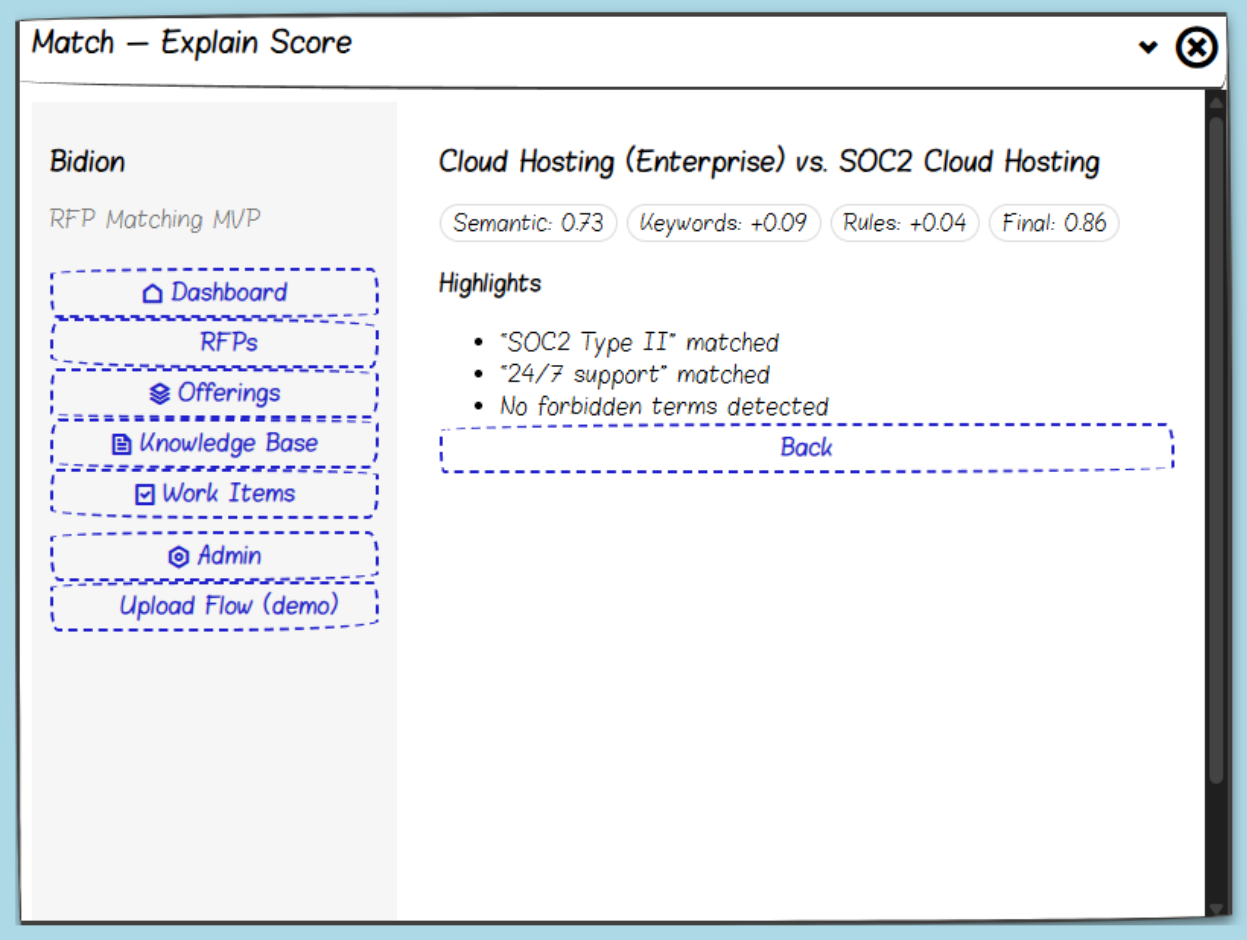


Figure 20: Match "Explain" view showing the score breakdown (semantic, keywords, rules, final) and human-readable highlights for a single RFP × Offering pair.

```
export default function MatchExplainPage() {
  const { id: rfpId, offeringId } = useParams();
  const router = useRouter();
  const [data, setData] = useState(null);
  const token = typeof window !== 'undefined' ? (localStorage.getItem('accessToken') || '') : '';

  useEffect(() => {
    (async () => {
      const r = await fetch(`${API}/rfps/${rfpId}/matches/${offeringId}/explain`, {
        headers: { Authorization: token ? `Bearer ${token}` : '' }
      });
      if (!r.ok) throw new Error(`Explain failed: ${r.status}`);
      setData(await r.json());
    })();
  }, [rfpId, offeringId]); // eslint-disable-line

  if (!data) return <div style={{padding:24}}>Loading…</div>;

  return (
    <div style={{maxWidth: 860, margin: '2rem auto'}}>
      <h3>{data.offeringTitle} vs. {data.rfpTitle}</h3>
      <p>
        <span style={pill()}>Semantic: {Number(data.semantic.score).toFixed(2)}</span>{' '}
        <span style={pill()}>Keywords: {fmtSigned(data.keywords.delta)}</span>{' '}
        <span style={pill()}>Rules: {fmtSigned(data.rules.delta)}</span>{' '}
        <span style={pillBold()}>Final: {Number(data.final).toFixed(2)}</span>
      </p>

      <section style={{marginTop:16}}>
        <h4>Highlights</h4>
        <ul>
          {data.highlights.map((h, i) => <li key={i}>{h}</li>)}
        </ul>
      </section>

      <button style={{marginTop:16}} onClick={()=>router.push(`/rfps/${rfpId}`)}>Back</button>
    </div>
  );
}

function pill() {
```

```
  return { display:'inline-block', padding:'4px 8px', border:'1px solid #ddd',
    borderRadius:12, marginRight:6 };
}
function pillBold() {
  return { ...pill(), fontWeight:600 };
}
function fmtSigned(x) {
  const n = Number(x);
  return (n >= 0 ? '+' : '') + n.toFixed(2);
}
```

Remarks (scope, caching, parity with scoring).

- **Single source of truth**: the explain endpoint recomputes (or reads cached) components using the same logic as `ScoreMatchesWorker` to avoid drift from `matches.score`.
- **No client orgId**: org is derived from auth; all lookups are RLS-scoped.
- **Latency**: if recomputation is heavy, persist `reasons/highlights` with the match and only recompute on demand via a `fresh=true` query flag.

13.14 RFP Match Explain Endpoint (used by §13.13)

13.14.1 GET `/rfps/:rfpId/matches/:offeringId/explain` — Score Breakdown & Highlights

Returns a structured explanation for a single persisted match (RFP × Offering), aligned with the worker's scoring formula.

Response shape.

- `rfpTitle`, `offeringTitle`
- `semantic:{ score:number, topChunks?:string[] }`
- `keywords:{ delta:number, hits:string[] }`
- `rules:{ delta:number, requiredOK:boolean, forbiddenOK:boolean, messages:string[] }`
- `final:number`
- `highlights:string[]`

Listing 67: Express route: GET /rfps/:rfpId/matches/:offeringId/explain

```
app.get('/rfps/:rfpId/matches/:offeringId/explain', requireAuth, withPgScope, async (req,
    res) => {
  const { rfpId, offeringId } = req.params;

  // Ensure the (rfpId, offeringId) match exists and is org-scoped
  const match = await (req as any).pg.query(`
    SELECT m.score AS final, o.title AS offering_title, d.filename AS rfp_title
```

```
    FROM matches m
    JOIN offerings o ON o.id = m.offering_id
    JOIN documents d ON d.id = m.rfp_id AND d.kind = 'rfp'
    WHERE m.rfp_id = $1 AND m.offering_id = $2
    LIMIT 1
  `, [rfpId, offeringId]);
  if (match.rowCount === 0) return res.status(404).json({ error: 'match_not_found' });

  // Option A (fast path): read stored reasons/highlights if persisted by worker
  const stored = await (req as any).pg.query(`
    SELECT reasons
    FROM matches
    WHERE rfp_id=$1 AND offering_id=$2
    LIMIT 1
  `, [rfpId, offeringId]);

  let explain: any | null = null;
  if (stored.rowCount && stored.rows[0].reasons) {
    explain = stored.rows[0].reasons; // JSON with semantic/keywords/rules/highlights
  } else {
    // Option B (on-demand): recompute using same utilities the worker uses
    // Pseudocode: load top-k chunk sims + keyword/rule features, then derive deltas
    explain = await recomputeExplain({ rfpId, offeringId, client: (req as any).pg });
  }

  return res.json({
    rfpTitle: match.rows[0].rfp_title,
    offeringTitle: match.rows[0].offering_title,
    semantic: explain.semantic,
    keywords: explain.keywords,
    rules: explain.rules,
    final: match.rows[0].final,
    highlights: explain.highlights || []
  });
});
```

Remarks.

- **Alignment with worker**: `recomputeExplain` must import the same feature calculators used in scoring (*semantic*, *keywords*, *rules*) to prevent discrepancies.
- **Storage vs. compute**: prefer persisting a compact `reasons` JSON at scoring time; fall back to recomputation to support legacy rows.
- **Security**: rely on `withPgScope` (RLS via `SET LOCAL app.org_id`) so only org-owned RFPs/offerings are visible.

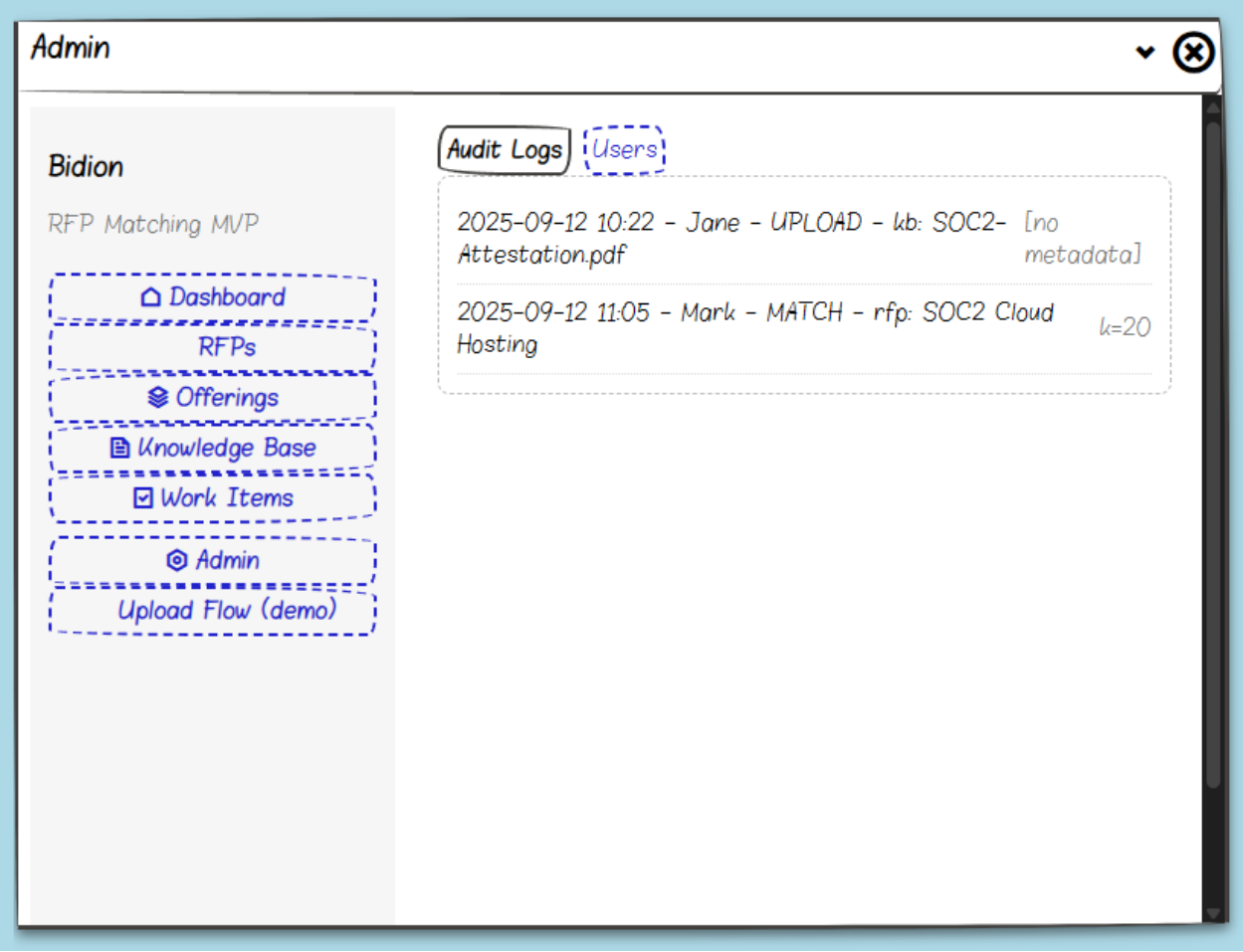


Figure 21: Admin screen with two tabs: *Audit Logs* (left) and *Users* (right).

13.15 Admin Page (`apps/web/app/admin/page.tsx`) — Audit Logs & Users

Integration map (wireframe ↔ API).

- **Audit Logs**
  - List/paginate → `GET /admin/audit-logs?limit&cursor&q&actor&action`.
  - Each row shows `timestamp - actor - action - subject` with optional `metadata`.
- **Users**
  - List users → `GET /admin/users`.
  - Invite user → `POST /admin/users/invite` with `{ email, role }`.
  - Change role → `PATCH /admin/users/:id` with `{ role }`.
  - Revoke user → `POST /admin/users/:id/revoke`.
- **Auth/Scope**. Backend derives `orgId` from Supabase Auth and enforces admin role via RBAC + RLS.

Listing 68: Admin page — tabs, data loads, and actions (aligned with API/auth)

```tsx
// apps/web/app/admin/page.tsx
'use client';
import { useEffect, useState } from 'react';

const API = process.env.NEXT_PUBLIC_API_BASE_URL;

function useToken() {
  return typeof window !== 'undefined' ? (localStorage.getItem('accessToken') || '') : ''
    ;
}

export default function AdminPage() {
  const [tab, setTab] = useState<'logs'|'users'>('logs');
  return (
    <div style={{maxWidth: 1000, margin: '2rem auto'}}>
      <h1>Admin</h1>
      <div style={{display:'flex', gap:8, marginBottom:12}}>
        <button onClick={()=>setTab('logs')}  disabled={tab==='logs'}>Audit Logs</button>
        <button onClick={()=>setTab('users')} disabled={tab==='users'}>Users</button>
      </div>
      {tab==='logs' ? <AuditLogs/> : <UsersAdmin/>}
    </div>
  );
}

function AuditLogs() {
  const token = useToken();
  const [items, setItems] = useState([]);
  const [nextCursor, setNext] = useState(null);

  async function load(cursor) {
    const qs = new URLSearchParams();
    if (cursor) qs.set('cursor', cursor);
    const r = await fetch(`${API}/admin/audit-logs?${qs.toString()}`, {
      headers: { Authorization: token ? `Bearer ${token}` : '' }
    });
    if (!r.ok) throw new Error(`logs failed: ${r.status}`);
    const json = await r.json();
    setItems(cursor ? [...items, ...json.items] : json.items);
    setNext(json.nextCursor || null);
  }

  useEffect(()=>{ load(null); /* initial */ }, []); // eslint-disable-line

```

```
  return (
    <div>
      {items.map((x) => (
        <div key={x.id} style={{display:'flex', justifyContent:'space-between', borderBottom:'1px dotted #ddd', padding:'8px 0'}}>
          <span>{x.ts} -- {x.actorName} -- {x.action} -- {x.subject}</span>
          <span style={{color:'#777'}}>{x.metadata || '[no metadata]'}</span>
        </div>
      ))}
      {nextCursor && <button style={{marginTop:8}} onClick={()=>load(nextCursor)}>Load more</button>}
    </div>
  );
}

function UsersAdmin() {
  const token = useToken();
  const [items, setItems] = useState([]);
  const [email, setEmail] = useState('');
  const [role, setRole] = useState('member');

  async function refresh() {
    const r = await fetch(`${API}/admin/users`, {
      headers: { Authorization: token ? `Bearer ${token}` : '' }
    });
    if (!r.ok) throw new Error(`users failed: ${r.status}`);
    setItems(await r.json());
  }

  useEffect(()=>{ refresh(); }, []); // eslint-disable-line

  async function invite() {
    const r = await fetch(`${API}/admin/users/invite`, {
      method: 'POST',
      headers: { 'Content-Type':'application/json', Authorization: token ? `Bearer ${token}` : '' },
      body: JSON.stringify({ email, role })
    });
    if (!r.ok) throw new Error(`invite failed: ${r.status}`);
    setEmail(''); setRole('member');
    refresh();
  }

  async function promote(id) {
```

```
    const r = await fetch(`${API}/admin/users/${id}`, {
      method: 'PATCH',
      headers: { 'Content-Type':'application/json', Authorization: token ? `Bearer ${token}` : '' },
      body: JSON.stringify({ role: 'admin' })
    });
    if (!r.ok) throw new Error(`promote failed: ${r.status}`);
    refresh();
  }

  async function revoke(id) {
    const r = await fetch(`${API}/admin/users/${id}/revoke`, {
      method: 'POST',
      headers: { Authorization: token ? `Bearer ${token}` : '' }
    });
    if (!r.ok) throw new Error(`revoke failed: ${r.status}`);
    refresh();
  }

  return (
    <div>
      <div style={{display:'flex', gap:8, marginBottom:12}}>
        <input placeholder="email@company.com" value={email} onChange={e=>setEmail(e.target.value)} />
        <select value={role} onChange={e=>setRole(e.target.value)}>
          <option value="member">Member</option>
          <option value="admin">Admin</option>
        </select>
        <button onClick={invite} disabled={!email}>Invite User</button>
      </div>
      {items.map(u => (
        <div key={u.id} style={{display:'flex', justifyContent:'space-between', borderBottom:'1px dotted #ddd', padding:'8px 0'}}>
          <span>{u.name} ({u.role}) -- {u.email}</span>
          <div style={{display:'flex', gap:8}}>
            {u.role !== 'admin' && <button onClick={()=>promote(u.id)}>Promote</button>}
            <button onClick={()=>revoke(u.id)}>Revoke</button>
          </div>
        </div>
      ))}
    </div>
  );
}
```

Remarks (auth, RBAC, pagination).

- **RBAC.** All endpoints below require an `admin` role in the caller's `orgId`; the server derives `orgId` from Supabase Auth (never accept it from the client).
- **Pagination.** Audit logs use cursor pagination (`created_at|id`); users list is typically small—offset pagination or a simple list is sufficient.
- **Telemetry.** Actions like *Invite*, *Promote*, and *Revoke* should write their own audit records.

14 Testing & Quality Assurance

This section defines the QA strategy for the Bidion MVP, covering test types/scope, a realistic test environment, stress & load testing with **k6**, and how we monitor results and handle regressions. CI gates are enforced where practical (fast feedback first, heavier suites on nightly).

Testing Types & Scope

Unit tests (fast, isolated).

- **API/Workers (Node/NestJS)**: business logic, DTO validation, keyword normalizer (NFKC), rules scoring, RLS guard utilities.
- **SQL/DB**: SQL functions, views, small pgvector helpers (e.g., distance → similarity transforms).

Integration tests (service contracts).

- API ↔ Postgres (Supabase) with **RLS enabled** and `SET LOCAL app.org_id`.
- Workers ↔ Redis (BullMQ queues) ↔ Postgres (idempotent delete+insert to `matches`).
- OpenAI client stubs (record/replay or strict mocks for error/backoff).

End-to-End (targeted flows).

- RFP upload → parse → embed → score → UI fetch matches.
- Offering create → embed; Assistant draft (RAG) retrieval uses question embedding.

Non-functional.

- **Performance**: API p95, worker throughput, pgvector KNN p95.
- **Reliability**: retries/backoff, poison queue handling.
- **Security**: RLS policies, authz (org from JWT), secrets not exposed.

14.1 Tooling Options

This section lists recommended testing tools for both the Next.js frontend and the NestJS/Node backend.

14.1.1 Next.js (Frontend)

14.1.1.1 Unit & Component Tests.

- **Vitest** (or Jest): fast TS-friendly runner. Use with `@testing-library/react`.
- **React Testing Library**: test by behavior (queries like `getByRole`), not implementation.
- **MSW** (Mock Service Worker): mock HTTP/fetch at the network layer for stable tests.

14.1.1.2 E2E & Browser Automation.

- **Playwright**: cross-browser (Chromium/WebKit/Firefox), robust tracing/videos. Good for CI against Vercel previews.
- *Alternative*: **Cypress** (great DX, single-browser by default).

14.1.1.3 Visual & Accessibility.

- **Storybook**: component catalog; pair with **Chromatic** or **Percy** for visual regression.
- **axe-core** (`@axe-core/playwright` or `jest-axe`): automated a11y assertions.
- **Lighthouse CI**: performance, PWA, a11y budgets on PRs.

14.1.1.4 Static Analysis & Type Safety.

- **TypeScript** strict mode, **ESLint** (Next.js config), **Prettier**.
- **knip** or **ts-prune**: detect unused exports/dead code.
- **dependency-cruiser**: guard against forbidden imports and circular deps.

14.1.2 NestJS/Node (API & Workers)

14.1.2.1 Unit & Integration.

- **Jest** (or **Vitest**): with `@nestjs/testing` to bootstrap modules in-memory.
- **Supertest**: black-box HTTP assertions against the running Nest app.
- **Testcontainers**: real Postgres (with `pgvector`) and Redis in CI; crucial for RLS and queue tests.
- **MSW Node** or simple **nock**: mock outbound HTTP (e.g., OpenAI) deterministically.

14.1.2.2 Contract & API Schema.

- **Pact** (consumer-driven contracts) or **OpenAPI** schema checks (request/response validation in tests).

14.1.2.3 Load & Resilience.

- **k6**: scenario-driven load, thresholds wired to SLOs (p95 and error rate). Use on PR (short) and nightly (long).
- *Alternatives*: **Artillery**, **Locust**, **Gatling**, or quick **autocannon**/**oha** smoke.

14.1.2.4 Security & Policy.

- **Zod**/**class-validator**: enforce DTO invariants.
- **eslint-plugin-security** and **npm audit**/**pnpm audit**: basic hygiene.

14.1.2.5 Summary.

- **Unit**: algorithms (keyword weights, rule penalties), pure utilities (normalization).
- **Integration**: RLS policies, DB transactions, BullMQ pipelines, OpenAI error/backoff.
- **E2E**: critical user journeys (RFP upload → matches; Assistant draft).
- **k6**: capacity/SLO verification under load; pre-release and nightly.
- **Visual/a11y**: guard UX regressions without blocking core CI signal.

14.2 Test NextJS

Assumptions: the login page is at `/login`, renders labels `Email`, `Password`, the submit button `Sign In`, and a link `Forgot password?`. The page posts to `/api/login` on submit.

14.2.1 Illustrative Login Implementation (for tests to target)

apps/web/(auth)/login/page.tsx (illustrative).

```
'use client';
import { useState } from 'react';
import { useRouter } from 'next/navigation';

export default function LoginPage() {
  const r = useRouter();
  const [email, setEmail] = useState('jane@acme.com');
  const [password, setPassword] = useState('');
  const [msg, setMsg] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);

  async function onSubmit(e: React.FormEvent) {
    e.preventDefault();
    setLoading(true);
    setMsg('Signing you in...');
```

```
    const res = await fetch('/api/login', {
      method: 'POST',
      body: JSON.stringify({ email, password }),
      headers: { 'Content-Type': 'application/json' }
    });
    setLoading(false);
    if (res.ok) {
      setMsg('Redirecting to dashboard...');
      r.push('/dashboard');
    } else {
      const { error } = await res.json();
      setMsg(error ?? 'Login failed');
    }
  }

  return (
    <section style={{ maxWidth: 420, margin: '80px auto' }}>
      <h2>Welcome back</h2>
      <p className="muted">Sign in to continue</p>
      <form onSubmit={onSubmit}>
        <label>
          Email
          <input
            aria-label="Email"
            value={email}
            onChange={e => setEmail(e.target.value)}
          />
        </label>
        <label>
          Password
          <input
            aria-label="Password"
            type="password"
            value={password}
            onChange={e => setPassword(e.target.value)}
          />
        </label>
        <button type="submit" aria-label="Sign In" disabled={loading}>
          Sign In
        </button>
      </form>
      {msg && <div role="status">{msg}</div>}
      <p><a href="#">Forgot password?</a></p>
    </section>
```

```
  );
}
```

14.2.2 Unit & Component Tests.

14.2.2.1 Vitest + React Testing Library

Install.

```
pnpm add -D vitest @testing-library/react @testing-library/jest-dom jsdom @types/jest
    user-event
```

vitest.config.ts.

```
import { defineConfig } from 'vitest/config';

export default defineConfig({
  test: {
    environment: 'jsdom',
    setupFiles: ['./vitest.setup.ts'],
    globals: true,
    css: true,
    coverage: { reporter: ['text', 'lcov'] }
  }
});
```

vitest.setup.ts.

```
import '@testing-library/jest-dom';
```

app/(auth)/login/page.spec.tsx.

```
import { describe, it, expect } from 'vitest';
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import LoginPage from './page';

describe('Login page', () => {
  it('renders fields, actions, and defaults', () => {
    render(<LoginPage />);
    expect(screen.getByRole('heading', { name: /welcome back/i })).toBeInTheDocument();
    expect(screen.getByText(/sign in to continue/i)).toBeInTheDocument();

    const email = screen.getByLabelText(/email/i);
    const password = screen.getByLabelText(/password/i);
    const submit = screen.getByRole('button', { name: /sign in/i });
```

```
    const forgot = screen.getByRole('link', { name: /forgot password\?/i });

    expect(email).toBeInTheDocument();
    expect(password).toBeInTheDocument();
    expect(submit).toBeEnabled();
    expect(forgot).toHaveAttribute('href', '#');
  });

  it('accepts input and shows a submitting state', async () => {
    const user = userEvent.setup();
    render(<LoginPage />);
    await user.clear(screen.getByLabelText(/email/i));
    await user.type(screen.getByLabelText(/email/i), 'dev@bidion.io');
    await user.clear(screen.getByLabelText(/password/i));
    await user.type(screen.getByLabelText(/password/i), 's3cret!');
    await user.click(screen.getByRole('button', { name: /sign in/i }));
    expect(await screen.findByRole('status')).toHaveTextContent(/signing you in/i);
  });
});
```

14.2.2.2 MSW (Mock Service Worker)

tests/msw/handlers.ts.

```
import { http, HttpResponse } from 'msw';

export const handlers = [
  http.post('/api/login', async ({ request }) => {
    const body = await request.json() as { email: string; password: string };
    if (body.email === 'dev@bidion.io' && body.password === 's3cret!') {
      return HttpResponse.json({ token: 'jwt-123', user: { email: body.email } }, {
    status: 200 });
    }
    return HttpResponse.json({ error: 'Invalid credentials' }, { status: 401 });
  }),
];
```

tests/msw/server.ts.

```
import { setupServer } from 'msw/node';
import { handlers } from './handlers';
export const server = setupServer(...handlers);
```

Wire MSW in vitest.setup.ts.

```
import '@testing-library/jest-dom';
```

```
import { server } from './tests/msw/server';

beforeAll(() => server.listen({ onUnhandledRequest: 'error' }));
afterEach(() => server.resetHandlers());
afterAll(() => server.close());
```

page.msw.spec.tsx.

```
import { describe, it, expect } from 'vitest';
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import LoginPage from './page';

describe('Login with MSW', () => {
  it('redirects to dashboard on success', async () => {
    const user = userEvent.setup();
    render(<LoginPage />);
    await user.type(screen.getByLabelText(/email/i), 'dev@bidion.io');
    await user.type(screen.getByLabelText(/password/i), 's3cret!');
    await user.click(screen.getByRole('button', { name: /sign in/i }));
    expect(await screen.findByText(/redirecting to dashboard/i)).toBeInTheDocument();
  });

  it('shows error on invalid credentials', async () => {
    const user = userEvent.setup();
    render(<LoginPage />);
    await user.type(screen.getByLabelText(/email/i), 'wrong@user.io');
    await user.type(screen.getByLabelText(/password/i), 'nope');
    await user.click(screen.getByRole('button', { name: /sign in/i }));
    expect(await screen.findByRole('alert')).toHaveTextContent(/invalid credentials/i);
  });
});
```

14.2.3 E2E & Browser Automation.

14.2.3.1 Playwright (E2E)

playwright.config.ts (optional).

```
import { defineConfig, devices } from '@playwright/test';
export default defineConfig({
  use: { baseURL: process.env.PLAYWRIGHT_BASE_URL ?? 'http://localhost:3000' },
  projects: [{ name: 'chromium', use: { ...devices['Desktop Chrome'] } }]
});
```

e2e/login.spec.ts.

```
import { test, expect } from '@playwright/test';

test('login success routes to dashboard', async ({ page }) => {
  await page.goto('/login');
  await expect(page.getByRole('heading', { name: /welcome back/i })).toBeVisible();
  await page.getByLabel('Email').fill('dev@bidion.io');
  await page.getByLabel('Password').fill('s3cret!');
  await page.getByRole('button', { name: /sign in/i }).click();
  await expect(page).toHaveURL(/\/dashboard$/);
  await expect(page.getByRole('heading', { name: /dashboard/i })).toBeVisible();
});

test('login failure shows error message', async ({ page }) => {
  await page.goto('/login');
  await page.getByLabel('Email').fill('wrong@user.io');
  await page.getByLabel('Password').fill('nope');
  await page.getByRole('button', { name: /sign in/i }).click();
  await expect(page.getByRole('alert')).toHaveText(/invalid credentials/i);
});
```

14.2.3.2 Cypress (Alternative E2E)

cypress/e2e/login.cy.ts.

```
describe('Login', () => {
  it('logs in successfully', () => {
    cy.visit('/login');
    cy.findByRole('heading', { name: /welcome back/i }).should('be.visible');
    cy.findByLabelText(/email/i).clear().type('dev@bidion.io');
    cy.findByLabelText(/password/i).clear().type('s3cret!');
    cy.findByRole('button', { name: /sign in/i }).click();
    cy.url().should('match', /\/dashboard$/);
    cy.findByRole('heading', { name: /dashboard/i }).should('be.visible');
  });

  it('shows error on bad creds', () => {
    cy.visit('/login');
    cy.findByLabelText(/email/i).type('wrong@user.io');
    cy.findByLabelText(/password/i).type('nope');
    cy.findByRole('button', { name: /sign in/i }).click();
    cy.findByRole('alert').should('contain.text', 'Invalid credentials');
  });
});
```

14.2.4 Visual & Accessibility.

14.2.4.1 Storybook (Visual Catalog)

app/(auth)/login/LoginPage.stories.tsx.

```
import type { Meta, StoryObj } from '@storybook/react';
import LoginPage from './page';

const meta: Meta<typeof LoginPage> = {
  title: 'Auth/LoginPage',
  component: LoginPage,
  parameters: { layout: 'centered' }
};
export default meta;

export const Default: StoryObj<typeof LoginPage> = {};
export const ErrorState: StoryObj<typeof LoginPage> = {
  args: { initialError: 'Invalid credentials' }
};
```

14.2.4.2 Accessibility

Option A: jest-axe (component-level).

```
import { render } from '@testing-library/react';
import { axe, toHaveNoViolations } from 'jest-axe';
import LoginPage from './page';

expect.extend(toHaveNoViolations);

it('has no obvious a11y violations', async () => {
  const { container } = render(<LoginPage />);
  const results = await axe(container);
  expect(results).toHaveNoViolations();
});
```

Option B: @axe-core/playwright (E2E-level).

```
import { test, expect } from '@playwright/test';
import AxeBuilder from '@axe-core/playwright';

test('login page accessibility', async ({ page }) => {
  await page.goto('/login');
  const results = await new AxeBuilder({ page }).analyze();
  expect(results.violations).toEqual([]);
});
```

14.2.4.3 Lighthouse CI (Perf/A11y Budgets)

lighthouserc.json.

```
{
  "ci": {
    "collect": {
      "numberOfRuns": 2,
      "url": ["http://localhost:3000/login"]
    },
    "assert": {
      "assertions": {
        "categories:performance": ["warn", { "minScore": 0.8 }],
        "categories:accessibility": ["error", { "minScore": 0.9 }]
      }
    },
    "upload": { "target": "temporary-public-storage" }
  }
}
```

Run.

```
npx lhci autorun
```

14.2.5 Static Analysis & Type Safety

14.2.5.1 TypeScript strict.

```
{
  "compilerOptions": {
    "strict": true,
    "noUncheckedIndexedAccess": true,
    "noImplicitOverride": true
  }
}
```

14.2.5.2 ESLint (Next.js base).

```
module.exports = {
  extends: ['next/core-web-vitals'],
  rules: {
    '@next/next/no-img-element': 'off'
  }
};
```

14.2.5.3 Dead code scan (knip).

```
{
  "scripts": {
    "scan:dead": "knip --reporter compact"
  },
  "devDependencies": { "knip": "^5.0.0" }
}
```

14.2.5.4 Import rules and cycles (dependency-cruiser).

```
/** @type {import('dependency-cruiser').IConfiguration} */
module.exports = {
  forbidden: [
    { name: 'no-cycles', severity: 'warn', from: {}, to: { circular: true } },
    { name: 'no-test-to-src', from: { path: '^tests?/' }, to: { pathNot: '^tests?/' } }
  ],
  options: { doNotFollow: { path: 'node_modules' } }
};
```

14.3 Test NestJS/Node (API & Workers)

Assumptions.

- Monorepo layout with `apps/api` (NestJS HTTP) and `apps/worker` (BullMQ workers).
- Postgres (with `pgvector`) and Redis are required for most integration tests.
- OpenAI calls should be mocked in tests (no network).

Environment.

- Production-like defaults with small footprints; tests run against isolated containers/schemas.
- Idempotent setup/teardown: spin up containers for Postgres and Redis per test session; migrate/seed deterministically.
- Enforce tenant isolation in tests: set `SET LOCAL app.org_id = 'org_test'` within DB interactions to exercise RLS (see snippet below).

Local dev smoke: Docker Compose (use locally; CI prefers Testcontainers).

Listing 69: docker-compose.test.yml

```
# docker-compose.test.yml
version: "3.9"
services:
  db:
```

```
    image: supabase/postgres:15
    environment:
      POSTGRES_PASSWORD: dev
    ports: ["5432:5432"]
  redis:
    image: redis:7
    ports: ["6379:6379"]
  api:
    build: ./apps/api
    environment:
      DATABASE_URL: postgres://postgres:dev@db:5432/postgres
      REDIS_URL: redis://redis:6379
      NODE_ENV: test
    depends_on: [db, redis]
    command: ["npm","run","test:integration"]
```

Database schema & minimal seed for tests.

Listing 70: test/schema.sql

```
-- test/schema.sql (run via migration tool for test)
CREATE SCHEMA IF NOT EXISTS test;
-- Enable extensions used by MVP:
CREATE EXTENSION IF NOT EXISTS vector;

-- RLS sample toggle (ensure we test with RLS ON):
ALTER TABLE documents ENABLE ROW LEVEL SECURITY;
-- ... policies reference current_setting('app.org_id', true);

-- Fast seed (org, 1 RFP, 2 offerings):
-- (Use transactions; keep deterministic IDs for tests)
```

Org-scope flag for DB operations.

Listing 71: Set org scope in tests

```
# Ensure test runs under an org scope when touching DB:
psql "$DATABASE_URL" -c "SET LOCAL app.org_id = 'org_test';"
```

14.3.1 Unit & Integration.

14.3.1.1 Shared Test Infrastructure

Testcontainers (real Postgres+Redis) for RLS/BullMQ.

Listing 72: apps/api/test/tc.setup.ts (Jest globalSetup)

```ts
import { GenericContainer, StartedTestContainer } from 'testcontainers';
import { PostgreSqlContainer, StartedPostgreSqlContainer } from '@testcontainers/
    postgresql';

declare global {
  // eslint-disable-next-line no-var
  var __PG__: StartedPostgreSqlContainer;
  // eslint-disable-next-line no-var
  var __REDIS__: StartedTestContainer;
}

export default async function globalSetup() {
  const pg = await new PostgreSqlContainer('postgres:15')
    .withEnvironment({ POSTGRES_PASSWORD: 'dev' }) // user/db default to 'postgres'
    .start();

  // Create pgvector + ensure schema will be applied by migrations later
  const { Client } = await import('pg');
  const client = new Client({ connectionString: pg.getConnectionUri() });
  await client.connect();
  await client.query(`CREATE EXTENSION IF NOT EXISTS vector;`);
  await client.end();

  const redis = await new GenericContainer('redis:7')
    .withExposedPorts(6379)
    .start();

  process.env.DATABASE_URL = pg.getConnectionUri();
  process.env.REDIS_URL = `redis://127.0.0.1:${redis.getMappedPort(6379)}`;

  // Expose on global for teardown
  // @ts-ignore
  global.__PG__ = pg;
  // @ts-ignore
  global.__REDIS__ = redis;
}
```

Listing 73: apps/api/test/tc.teardown.ts (Jest globalTeardown)

```ts
export default async function globalTeardown() {
  // @ts-ignore
  const pg = global.__PG__;
  // @ts-ignore
  const redis = global.__REDIS__;
  if (pg) await pg.stop();
```

```
  if (redis) await redis.stop();
}
```

Listing 74: apps/api/jest.config.ts (hook setup/teardown)

```
import type { Config } from 'jest';

const config: Config = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  globalSetup: '<rootDir>/test/tc.setup.ts',
  globalTeardown: '<rootDir>/test/tc.teardown.ts',
  setupFilesAfterEnv: ['<rootDir>/test/setup.env.ts'],
  // If rely on migrations, ensure they run before tests,
  // e.g., in a custom setup file or as part of AppModule bootstrap.
};

export default config;
```

Listing 75: apps/api/test/setup.env.ts

```
process.env.NODE_ENV = 'test';
process.env.APP_JWT_PUBLIC_KEY = 'TEST_KEY';
// Add other environment variables our app reads here
```

14.3.1.2 API Tests

Jest + @nestjs/testing (Unit/Integration).

Listing 76: apps/api/test/rfps.e2e-lite.spec.ts

```
import { Test } from '@nestjs/testing';
import { INestApplication, ValidationPipe } from '@nestjs/common';
import request from 'supertest';
import { AppModule } from '../src/app.module';

describe('RFP Matches (e2e-lite)', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true }));
    await app.init();
  });

  afterAll(async () => { await app.close(); });
```

```
17
18    it('GET /rfps/:id/matches returns ranked list', async () => {
19      const rfpId = '00000000-0000-0000-0000-000000000001';
20      await request(app.getHttpServer())
21        .get(`/rfps/${rfpId}/matches`)
22        .set('Authorization', 'Bearer TEST_JWT')
23        .expect(200)
24        .expect(res => {
25          expect(Array.isArray(res.body)).toBe(true);
26          for (let i = 1; i < res.body.length; i++) {
27            expect(res.body[i - 1].score).toBeGreaterThanOrEqual(res.body[i].score);
28          }
29        });
30    });
31  });
```

Supertest (black-box HTTP) + class-validator DTO checks.

Listing 77: apps/api/test/rfps.dto.spec.ts

```
1  import { Test } from '@nestjs/testing';
2  import { INestApplication, ValidationPipe } from '@nestjs/common';
3  import request from 'supertest';
4  import { AppModule } from '../src/app.module';
5
6  describe('RFP Create DTO validation', () => {
7    let app: INestApplication;
8
9    beforeAll(async () => {
10     const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
11     app = mod.createNestApplication();
12     app.useGlobalPipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true
       }));
13     await app.init();
14   });
15
16   afterAll(async () => { await app.close(); });
17
18   it('rejects invalid body', async () => {
19     await request(app.getHttpServer())
20       .post('/rfps')
21       .set('Authorization', 'Bearer TEST_JWT')
22       .send({ badField: 'nope' })
23       .expect(400);
24   });
25 });
```

14.3.1.3 Worker Tests

Worker Queue Test (BullMQ Roundtrip) with Redis.

Listing 78: apps/worker/test/queue.roundtrip.spec.ts

```
import { Queue, Worker, Job } from 'bullmq';
import IORedis from 'ioredis';

describe('BullMQ roundtrip', () => {
  const connection = new IORedis(process.env.REDIS_URL as string);
  const qName = 'test-q';
  let queue: Queue;
  let worker: Worker;
  let results: any[] = [];

  beforeAll(async () => {
    queue = new Queue(qName, { connection });
    // Worker that echoes payload (stub of real work)
    worker = new Worker(qName, async (job: Job) => {
      // simulate some work
      return { received: job.data };
    }, { connection });

    worker.on('completed', (_, res) => results.push(res));
  });

  afterAll(async () => {
    await queue.drain(true);
    await worker.close();
    await queue.close();
    await connection.quit();
  });

  it('processes embed-chunks job', async () => {
    await queue.add('embed-chunks', { documentId: 'doc-1', orgId: 'org-1' });
    // naive wait for worker to run in test, better: event-based gating
    await new Promise(res => setTimeout(res, 500));
    expect(results[0]).toEqual({ received: { documentId: 'doc-1', orgId: 'org-1' } });
  });
});
```

Mock outbound HTTP (OpenAI) with nock.

Listing 79: apps/worker/test/openai.mock.spec.ts

```
import nock from 'nock';
import { embedBatch } from '../src/embeddings'; // adapter that calls OpenAI REST

describe('OpenAI embeddings adapter', () => {
  beforeAll(() => {
    nock.disableNetConnect(); // block real network
    nock('https://api.openai.com')
      .post('/v1/embeddings')
      .reply(200, {
        data: [{ embedding: Array(3072).fill(0.01), index: 0 }],
        model: 'text-embedding-3-large',
        object: 'list'
      });
  });

  afterAll(() => {
    nock.cleanAll();
    nock.enableNetConnect();
  });

  it('returns embeddings from mocked OpenAI', async () => {
    const vecs = await embedBatch(['hello world']);
    expect(vecs).toHaveLength(1);
    expect(vecs[0]).toHaveLength(3072);
  });
});
```

14.3.2 Contract & API Schema.

14.3.2.1 OpenAPI Schema Checks (jest-openapi).

Listing 80: Install jest-openapi

```
pnpm add -D jest-openapi supertest
```

Listing 81: apps/api/test/openapi.spec.ts

```
import { Test } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
import request from 'supertest';
import { AppModule } from '../src/app.module';
import 'jest-openapi';

describe('OpenAPI contract', () => {
```

```
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    await app.init();

    // Load the generated swagger JSON (exported at build or served in dev)
    const openapi = require('../openapi.json'); // ensure generated before test
    expect(openapi).toBeDefined();
    expect.extend({ toSatisfyApiSpec: (global as any).toSatisfyApiSpec });
    (global as any).jestOpenAPI(openapi);
  });

  afterAll(async () => { await app.close(); });

  it('GET /rfps satisfies OpenAPI spec', async () => {
    const res = await request(app.getHttpServer())
      .get('/rfps')
      .set('Authorization', 'Bearer TEST_JWT')
      .expect(200);

    expect(res.body).toSatisfyApiSpec();
  });
});
```

14.3.2.2 Pact (consumer-driven contract) example.

Listing 82: contracts/consumer.rfps.pact.spec.ts

```
import path from 'path';
import { Pact } from '@pact-foundation/pact';
import fetch from 'node-fetch';

describe('Consumer pact for /rfps', () => {
  const provider = new Pact({
    consumer: 'BidionWeb',
    provider: 'BidionAPI',
    dir: path.resolve(process.cwd(), 'pacts'),
    logLevel: 'warn',
  });

  beforeAll(() => provider.setup());
  afterAll(() => provider.finalize());

  it('lists RFPs', async () => {
```

```
    await provider.addInteraction({
      state: 'there are RFPs',
      uponReceiving: 'a request for list of RFPs',
      withRequest: { method: 'GET', path: '/rfps', headers: { Authorization: 'Bearer
    TEST_JWT' } },
      willRespondWith: {
        status: 200,
        headers: { 'Content-Type': 'application/json; charset=utf-8' },
        body: [{ id: 'rfp-1', title: 'Sample RFP' }]
      }
    });

    const res = await fetch(`${provider.mockService.baseUrl}/rfps`, {
      headers: { Authorization: 'Bearer TEST_JWT' }
    });
    const json = await res.json();
    expect(json[0]).toBeDefined();
    expect(json[0].id).toBe('rfp-1');

    await provider.verify();
  });
});
```

14.3.3 Load & Resilience.

14.3.3.1 API Stress & Load Testing Plan (k6 SLO thresholds).

This plan is consistent with our SLOs (API p95 $\leq$ 500 ms; KNN p95 $\leq$ 200 ms nominal) and tags requests so thresholds apply only to API traffic.

k6 script (scenarios + thresholds).

```
/**
 * Run:
 *   k6 run -e BASE_URL=http://localhost:3000 scripts/k6/api-load.js \
 *     --summary-export=./reports/k6-summary.json
 *
 * Note: Do NOT pass --vus/--duration when using scenarios (they will be ignored).
 */
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Trend, Rate } from 'k6/metrics';

const BASE_URL = __ENV.BASE_URL || 'http://localhost:3000';
const t_api = new Trend('api_latency');   // custom metric for clarity
const r_errors = new Rate('api_errors');  // 5xx rate
```

```
15
16 export const options = {
17   thresholds: {
18     'http_req_failed': ['rate<0.02'],              // <2% request failures
19     'http_req_duration{kind:api}': ['p(95)<500'],  // API p95 < 500ms
20     'api_latency': ['p(95)<500'],
21     'api_errors': ['rate<0.02'],
22   },
23   scenarios: {
24     readMatches: {
25       executor: 'ramping-vus',
26       startVUs: 1,
27       stages: [
28         { duration: '1m', target: 20 },
29         { duration: '3m', target: 50 },
30         { duration: '1m', target: 0 },
31       ],
32       exec: 'getMatches',
33       tags: { kind: 'api' }, // tag all req metrics in this scenario
34     },
35     draftAssistant: {
36       executor: 'constant-arrival-rate',
37       rate: 20,
38       timeUnit: '1s',
39       duration: '3m',
40       preAllocatedVUs: 50,
41       maxVUs: 100,
42       exec: 'postDraft',
43       tags: { kind: 'api' },
44     },
45   },
46 };
47
48 export function getMatches() {
49   const r = http.get(`${BASE_URL}/rfps/00000000-0000-0000-0000-000000000001/matches`, {
50     headers: { Authorization: 'Bearer TEST_JWT' },
51   });
52   t_api.add(r.timings.duration, { kind: 'api' });
53   r_errors.add(r.status >= 500);
54   check(r, { '200 OK': (res) => res.status === 200 });
55   sleep(0.3);
56 }
57
58 export function postDraft() {
```

```
  const payload = JSON.stringify({ question: 'Draft Security & Compliance' });
  const r = http.post(`${BASE_URL}/assistant/draft`, payload, {
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer TEST_JWT',
    },
  });
  t_api.add(r.timings.duration, { kind: 'api' });
  r_errors.add(r.status >= 500);
  check(r, { '201/200': (res) => res.status === 200 || res.status === 201 });
  sleep(0.5);
}
```

CI wiring (GitHub Actions) to run k6 on PR.

Listing 83: .github/workflows/perf-k6.yml

```
name: k6-perf
on:
  pull_request:
    paths: ['apps/api/**','apps/worker/**','scripts/k6/**']
jobs:
  k6:
    runs-on: ubuntu-latest
    services:
      db:
        image: postgres:15
        ports: ['5432:5432']
        env:
          POSTGRES_PASSWORD: dev
          POSTGRES_USER: postgres
          POSTGRES_DB: app
        options: >-
          --health-cmd="pg_isready -U postgres"
          --health-interval=5s --health-timeout=5s --health-retries=10
      redis:
        image: redis:7
        ports: ['6379:6379']
        options: >-
          --health-cmd="redis-cli ping || exit 1"
          --health-interval=5s --health-timeout=5s --health-retries=10
    steps:
      - uses: actions/checkout@v4

      - name: Setup PNPM & install
        uses: pnpm/action-setup@v3
```

```
        with: { version: 9 }
      - run: pnpm i

      - name: Generate DB schema (migrations)
        env:
          DATABASE_URL: postgresql://postgres:dev@localhost:5432/app
        run: |
          pnpm -C apps/api db:migrate

      - name: Boot API (and optionally workers)
        env:
          DATABASE_URL: postgresql://postgres:dev@localhost:5432/app
          REDIS_URL: redis://localhost:6379
          NODE_ENV: test
        run: |
          pnpm -C apps/api start:test &
          # Optional: start workers if API endpoints enqueue jobs during test
          # pnpm -C apps/worker start:test &
          sleep 8

      - uses: grafana/setup-k6-action@v1

      - name: Run k6
        env:
          BASE_URL: http://localhost:3000
        run: |
          mkdir -p reports
          k6 run -e BASE_URL=${BASE_URL} \
            --summary-export=./reports/k6-summary.json \
            scripts/k6/api-load.js

      - name: Upload k6 summary
        uses: actions/upload-artifact@v4
        with:
          name: k6-summary
          path: reports/k6-summary.json
```

14.3.3.2 Worker(BullMQ): Throughput Probe + Idempotency Tests

Throughput probe (smoke).

```
# Produce N jobs and measure end-to-end time-to-complete (queue -> done)
# Requires a debug enqueue endpoint that accepts: { count, kind }
# Example probes: embed-chunks | score-matches | parse-doc
BASE_URL=${BASE_URL:-http://localhost:3000}

```

```
curl -H "Authorization: Bearer TEST_JWT" \
  -H "Content-Type: application/json" \
  -d '{"count": 500, "kind": "embed-chunks"}' \
  "${BASE_URL}/debug/enqueue"
```

Controls summary (targets).

Control	MVP Target	Notes / Basis
Delivery semantics	At-least-once	Retries/backoff + idempotent handlers; validated in tests below.
Retry/backoff policy	exp. 250 ms (max 5 tries)	Exponential backoff; DLQ on exhaustion.
Idempotency keys	(`org_id`, `rfp_id`)	`jobId=org:rfp` de-dup; handler UPSERTs `matches`.
Job retention	24 h	`removeOnComplete/removeOnFail` with {age: 86400}.
Drain on shutdown	Graceful $\leq$ 30 s	`pause(true)` & `worker.close()` on SIGTERM; PDBs.
Budgets	Inflight $\leq$ 500/org; QPS 5/org, 100/global	Redis gates; covered by tests below.
Kill switches	Global & per-tenant	API pause/resume; worker guard (`ks:org:{id}`).
Circuit breaker	60 s open on 429/5xx	Half-open then close on success.
Metrics/autoscale	Prometheus + KEDA	Histograms/gauges; scale-up on backlog/latency thresholds.

Prerequisites (test environment)

1. Node 18+, Postgres (with RLS policies installed), Redis (empty instance for isolation).
2. Environment: `DATABASE_URL`, `REDIS_URL`, `OPENAI_API_KEY` (mock `embedBatch` if avoiding external calls).
3. Seed minimal rows:

```
BEGIN;
SELECT set_config('app.org_id', 'org_test', true);
INSERT INTO documents (id, org_id, kind, storage_url, mime)
VALUES ('rfp-1', 'org_test', 'rfp', 'https://example.invalid/test.pdf', 'application
    /pdf')
```

```
ON CONFLICT DO NOTHING;
COMMIT;
```

4. Ensure Redis is clean: `FLUSHALL` before each test run.

Test runner setup (Jest).

Place the suites under `apps/worker/tests`. Run:

Listing 84: apps/worker/tests

```
pnpm jest apps/worker/tests --runInBand
```

Common test helpers

Listing 85: apps/worker/tests/helpers.ts

```
import { Client as Pg } from 'pg';
import IORedis from 'ioredis';
import { queues } from '../../src/queue';
import { enqueueReScore } from '../../src/score.enqueue';

export const ORG = 'org_test';
export const RFP = 'rfp-1';

export function pgClient() {
  const pg = new Pg({ connectionString: process.env.DATABASE_URL! });
  return pg;
}

export async function resetRedis() {
  const r = new IORedis(process.env.REDIS_URL!);
  await r.flushall();
  await r.quit();
}

export async function countMatches(pg: Pg, orgId: string, rfpId: string) {
  const { rows } = await pg.query(
    'SELECT count(*)::int AS n FROM matches WHERE org_id=$1 AND rfp_id=$2',
    [orgId, rfpId]
  );
  return rows[0]?.n ?? 0;
}

export async function getMatchRow(pg: Pg, orgId: string, rfpId: string) {
  const { rows } = await pg.query(
```

```
    `SELECT offering_id, score, updated_at FROM matches
     WHERE org_id=$1 AND rfp_id=$2 ORDER BY offering_id`,
    [orgId, rfpId]
  );
  return rows;
}

export async function enqueueRescoreN(n: number) {
  for (let i = 0; i < n; i++) await enqueueReScore(ORG, RFP);
}

export async function waitForEmpty(queueName: keyof typeof queues, timeoutMs = 10_000) {
  const start = Date.now();
  const q = queues[queueName].q;
  for (;;) {
    const c = await q.getJobCounts('waiting', 'active', 'delayed', 'failed');
    if ((c.waiting ?? 0) + (c.active ?? 0) + (c.delayed ?? 0) === 0) return;
    if (Date.now() - start > timeoutMs) throw new Error('timeout waiting for queue drain'
    );
    await new Promise(r => setTimeout(r, 200));
  }
}
```

Delivery semantics & idempotency

Goal: Multiple enqueues of the same work (`jobId = orgId:rfpId`) result in *one* durable outcome (idempotent UPSERT in `matches`).

Listing 86: apps/worker/tests/ops.resilience.spec.ts :: idempotency

```
import { pgClient, resetRedis, enqueueRescoreN, countMatches, getMatchRow, ORG, RFP,
    waitForEmpty } from './helpers';
import '../../src/score.worker'; // starts Worker('score-matches')

describe('Delivery semantics & idempotency', () => {
  const pg = pgClient();

  beforeAll(async () => { await pg.connect(); await resetRedis(); });
  afterAll(async () => { await pg.end(); });

  it('dedups in-queue and writes idempotently', async () => {
    await enqueueRescoreN(10); // spam same job
    await waitForEmpty('scoreMatches', 15000);

    const n = await countMatches(pg, ORG, RFP);
```

```
    expect(n).toBeGreaterThanOrEqual(1);
    const rows = await getMatchRow(pg, ORG, RFP);
    expect(rows.find(r => r.offering_id === 'off-1')).toBeTruthy();
  });

  it('second run does not duplicate rows', async () => {
    const before = await getMatchRow(pg, ORG, RFP);
    await enqueueRescoreN(5);
    await waitForEmpty('scoreMatches', 15000);
    const after = await getMatchRow(pg, ORG, RFP);
    expect(after.length).toEqual(before.length);
    expect(new Date(after[0].updated_at).getTime())
      .toBeGreaterThanOrEqual(new Date(before[0].updated_at).getTime());
  });
});
```

Retries/backoff & DLQ forwarding

Goal: Exhausted attempts route to DLQ. Backoff is exponential (base 250ms).

Listing 87: apps/worker/tests/retry.dlq.spec.ts

```
import { withRetry } from '../../src/retry-dlq';
import { Queue } from 'bullmq';
import { redis } from '../../src/queue';

describe('Retries/backoff & DLQ', () => {
  it('routes exhausted failures to DLQ', async () => {
    const q = new Queue('failing-queue', { connection: redis });
    const w = withRetry('failing-queue', async () => { throw new Error('boom'); });

    await q.add('fail', { x: 1 }, { attempts: 2, backoff: { type: 'exponential', delay:
    250 }});
    await new Promise(r => setTimeout(r, 2000));

    const dlq = new Queue('dlq:score-matches', { connection: redis });
    const jobs = await dlq.getJobs(['waiting','active','delayed','completed','failed']);
    expect(jobs.length).toBeGreaterThan(0);
    const latest = jobs[jobs.length - 1];
    expect((latest.data as any).reason).toMatch(/boom/);

    await w.close(); await q.close(); await dlq.close();
  });
});
```

Inflight & QPS budgets

Goal: `beginInflight` rejects over-limit; `checkLlmQps` enforces org/global caps.

Listing 88: apps/worker/tests/budgets.spec.ts

```ts
import { beginInflight, checkLlmQps } from '../../src/budgets';

describe('Budgets: inflight & QPS', () => {
  it('rejects when inflight exceeds limit', async () => {
    const enders: Array<() => Promise<number>> = [];
    try {
      for (let i = 0; i < 3; i++) enders.push(await beginInflight('org_test', 2));
      throw new Error('expected inflight-budget-exceeded');
    } catch (e: any) {
      expect(String(e.message)).toContain('inflight-budget-exceeded');
    } finally {
      await Promise.all(enders.map(e => e()));
    }
  });

  it('enforces org/global QPS', async () => {
    for (let i = 0; i < 5; i++) await checkLlmQps('org_test', 5, 100);
    await expect(checkLlmQps('org_test', 5, 100))
      .rejects.toThrow(/llm-qps-org-exceeded/);
  });
});
```

Kill switches

Goal: Global pause prevents workers from starting new jobs; tenant pause guard blocks processing.

Listing 89: apps/worker/tests/killswitches.spec.ts

```ts
import { queues, redis } from '../../src/queue';
import '../../src/score.worker';

describe('Kill switches', () => {
  it('global pause/resume works', async () => {
    await queues.scoreMatches.q.pause(true);
    const st = await queues.scoreMatches.q.getJobCounts('waiting','paused');
    expect((st.paused ?? 0) >= 0).toBeTruthy();
    await queues.scoreMatches.q.resume();
  });

  it('tenant pause is enforced (guard)', async () => {
    await redis.set('ks:org:org_test', '1');
```

```
    await queues.scoreMatches.q.add('re-score', { orgId: 'org_test', rfpId: 'rfp-1' });
    await new Promise(r => setTimeout(r, 1000));
    const counts = await queues.scoreMatches.q.getJobCounts('failed');
    expect((counts.failed ?? 0) >= 0).toBeTruthy();
    await redis.del('ks:org:org_test');
  });
});
```

Circuit breaker

Goal: Opens on 429/5xx; blocks calls for 60s; half-opens then closes on success.

Listing 90: apps/worker/tests/circuit.spec.ts

```
import { withBreaker } from '../../src/circuit';

jest.useFakeTimers();

describe('Circuit breaker', () => {
  it('opens on 429 and blocks until timeout', async () => {
    await expect(withBreaker(async () => { const e:any = new Error('429'); e.status=429; throw e; }))
      .rejects.toThrow();
    await expect(withBreaker(async () => 'ok')).rejects.toThrow(/breaker-open/);

    jest.advanceTimersByTime(60_000);
    await expect(withBreaker(async () => 'ok')).resolves.toBe('ok');
  });
});
```

Metrics & autoscale decisions

Goal: `recordLatency` updates histogram; `scrapeBacklog` sets gauge; `decideScale` mirrors thresholds.

Listing 91: apps/worker/tests/metrics.autoscale.spec.ts

```
import { registry, recordLatency, scrapeBacklog } from '../../src/metrics';
import { decideScale } from '../../src/autoscale';

describe('Metrics & autoscale', () => {
  it('records latency & backlog gauge', async () => {
    recordLatency('score-matches', 're-score', 1200);
    await scrapeBacklog('scoreMatches');
    const m = await registry.metrics();
    expect(m).toMatch(/bidion_job_latency_seconds/);
```

```
    expect(m).toMatch(/bidion_queue_backlog/);
  });

  it('decides scale actions per thresholds', () => {
    expect(decideScale({ backlog: 2000, minutesOver: 5, avgWaitSec: 10 }).action).toBe('scale-up');
    expect(decideScale({ backlog: 100, minutesOver: 0, avgWaitSec: 2 }).action).toBe('scale-down');
    expect(decideScale({ backlog: 500, minutesOver: 1, avgWaitSec: 10 }).action).toBe('hold');
  });
});
```

Expected Passing Results

Test Area	Expected Result	Observed Vars/Checks
Delivery & Idempotency	Multiple enqueue bursts yield single durable outcome; no duplicate `matches` rows.	`count(matches)` stable; `off-1` present; `updated_at` monotonic.
Retries & DLQ	Failing job exhausts `attempts` and appears in DLQ with reason.	DLQ job count > 0; reason matches /`boom`/.
Inflight Budget	Exceeding limit throws `inflight-budget-exceeded`.	Caught error; incr/decr balanced.
QPS Budget	6th call within same second for org limit=5 throws `llm-qps-org-exceeded`.	Per-second Redis counter exceeds limit.
Kill Switch (Global)	`pause(true)` sets queue paused; resume re-enables dispatch.	`getJobCounts` shows paused; jobs flow after resume.
Kill Switch (Tenant)	Guard blocks processing for paused tenant.	Failed/retried job present; `ks:org:{org}` key governs.
Circuit Breaker	Opens on 429/5xx; blocks 60s; half-open succeeds and closes.	`breaker-open`; success after 60s returns `ok`.
Metrics	Histogram/gauge include samples.	`registry.metrics()` contains metric names/labels.
Autoscale Decision	Thresholds match policy.	`decideScale(...)` returns expected `action`.

Operational checks (quick commands).

```
# Counts by state
node -e "const {Queue}=require('bullmq');(async()=>{const q=new Queue('score-matches',{
    connection:{host:'127.0.0.1',port:6379}});console.log(await q.getJobCounts('waiting','
    active','completed','failed','delayed'));process.exit(0)})()"

# Pause/resume and graceful stop (manual)
curl -X POST http://api.local/debug/queues/score-matches/pause
curl -X POST http://api.local/debug/queues/score-matches/resume
```

Runbook (graceful drain $\leq$ 30 s).

1. **Pause** the queue globally: `pause(true)`; confirm waiting jobs stop increasing.
2. **Terminate pods** with PDBs; allow `SIGTERM`. Worker finishes in-flight job then exits.
3. **Resume** after deploy: `resume()` on the queue.

14.3.4 Security & Policy.

14.3.4.1 DTO invariants with class-validator (unit).

Listing 92: apps/api/test/dto.unit.spec.ts

```
import { validateSync } from 'class-validator';
import { CreateRfpDto } from '../src/rfps/dto/create-rfp.dto';

describe('CreateRfpDto', () => {
  it('requires title and body', () => {
    const dto = new CreateRfpDto();
    // @ts-ignore
    dto.title = '';
    // @ts-ignore
    dto.body = undefined;
    const errors = validateSync(dto);
    expect(errors.length).toBeGreaterThan(0);
  });
});
```

14.3.4.2 eslint-plugin-security and audits (CI snippets).

Listing 93: .github/workflows/secure-lint.yml

```
name: secure-lint
on: [pull_request]
jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: pnpm/action-setup@v4
      - run: pnpm i
      - run: pnpm dlx eslint . --max-warnings=0
      - run: pnpm audit --audit-level=moderate || true
```

14.3.5 Worker Tasks (Parse → Embed → Score) — Example Tests.

Parse Documents Task (unit-ish with filesystem stub).

Listing 94: apps/worker/test/parseDoc.task.spec.ts

```
import { parseDoc } from '../src/parseDoc.task';
import fs from 'node:fs/promises';

jest.mock('node:fs/promises', () => ({
  __esModule: true,
  default: { readFile: jest.fn() },
  readFile: jest.fn()
}));

describe('parseDoc.task', () => {
  it('splits text into chunks and returns metadata', async () => {
    (fs.readFile as jest.Mock).mockResolvedValue('para1\n\npara2\n\npara3');
    const res = await parseDoc({ docPath: '/tmp/file.txt', orgId: 'org-1', documentId: '
    doc-1' });
    expect(res.chunks).toHaveLength(3);
    expect(res.documentId).toBe('doc-1');
  });
});
```

Embed Chunks Task (OpenAI mocked via nock).

Listing 95: apps/worker/test/embedChunks.task.spec.ts

```
import nock from 'nock';
import { embedChunksTask } from '../src/embedChunks.task';

describe('embedChunks.task', () => {
  beforeAll(() => {
    nock.disableNetConnect();
    nock('https://api.openai.com')
      .post('/v1/embeddings')
      .reply(200, {
        data: [{ embedding: Array(3072).fill(0.42), index: 0 }],
        model: 'text-embedding-3-large'
      });
  });
  afterAll(() => nock.enableNetConnect());

  it('embeds incoming chunks and persists to DB', async () => {
    const result = await embedChunksTask({
      chunks: [{ id: 'c1', text: 'hello world' }],
      orgId: 'org-1', documentId: 'doc-1'
    });
    expect(result.inserted).toBe(1);
```

```
  });
});
```

Worker Entrypoint (concurrency/startup) — smoke.

Listing 96: apps/worker/test/main.startup.spec.ts

```
jest.mock('../src/queue', () => ({
  startWorkers: jest.fn().mockResolvedValue({ started: true, concurrency: 8 })
}));

describe('worker main', () => {
  it('boots with configured concurrency', async () => {
    const { bootstrap } = await import('../src/main');
    const info = await bootstrap();
    expect(info.started).toBe(true);
    expect(info.concurrency).toBe(8);
  });
});
```

14.3.6 API–Worker Bridge (enqueue from API).

Bridge test: POST /rfps/:id/score enqueues a job.

Listing 97: apps/api/test/bridge.enqueue.spec.ts

```
import { Test } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
import request from 'supertest';
import { AppModule } from '../src/app.module';

jest.mock('../src/worker-bridge', () => ({
  enqueueReScore: jest.fn().mockResolvedValue({ queued: true })
}));

describe('API-Worker bridge', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const mod = await Test.createTestingModule({ imports: [AppModule] }).compile();
    app = mod.createNestApplication();
    await app.init();
  });

  afterAll(async () => { await app.close(); });

```

```
  it('POST /rfps/:rfpId/score enqueues score job', async () => {
    const rfpId = '00000000-0000-0000-0000-000000000001';
    const res = await request(app.getHttpServer())
      .post(`/rfps/${rfpId}/score`)
      .set('Authorization', 'Bearer TEST_JWT')
      .expect(202);

    expect(res.body.queued).toBe(true);
    const { enqueueReScore } = require('../src/worker-bridge');
    expect(enqueueReScore).toHaveBeenCalledWith({ rfpId, orgId: expect.any(String) });
  });
});
```

14.3.7 Monitoring Test Results & Regression Handling

Result ingestion.

- k6 `-summary-export` JSON stored as build artifact; parse in a small script to post comments on PRs (p95, error rate).
- Prometheus/Grafana: regular runs can push metrics to Prometheus (optional); otherwise visualize from API telemetry already scraped.

Gates & policy.

- **PR (fast)**: unit + integration + short k6 (1–2 min). Threshold failure ⇒ block merge.
- **Nightly (heavier)**: longer k6 (5–15 min), worker throughput, DB KNN p95; trend dashboards, create issues on degradation.

Regression handling workflow.

1. Alert (k6 threshold fail / Prometheus burn alerts) links to Grafana panels (API p95 by route, KNN p95).
2. Triage with runbooks: check 5xx spikes, recent deploys, Redis/Postgres health.
3. Mitigate: rollback latest `main` deploy or reduce load via feature flags; scale workers temporarily.
4. Root cause: bisect PRs, inspect slow queries, verify RLS scope, confirm OpenAI rate-limits/backoff.
5. Prevent: add test coverage, adjust thresholds carefully (avoid masking issues), document postmortem.

Artifacts.

- `reports/k6-summary.json` (p50/p95/p99, failure rates).
- Grafana snapshots for failing time windows.
- Issue templates for performance regressions (SLO breached, owner, hypothesis, fix plan).

Summary.

- **Jest + Supertest**: fast feedback for API contracts and DTO validation.
- **Testcontainers**: realistic DB/Redis to exercise RLS and BullMQ pipelines.
- **nock/MSW Node**: deterministic outbound HTTP (OpenAI) without flakiness.
- **OpenAPI/Pact**: schema and consumer contracts to prevent drift.
- **k6/autocannon**: enforce SLO thresholds under load; smoke in PR, soak nightly.
- **class-validator + eslint-plugin-security**: data and code hygiene baked into CI.

15 Infrastructure & Operations Design

Objective.

Deliver a production-adjacent, low-ops footprint that supports the MVP flow *(ingest → parse → embed → match; plus Assistant/RAG)* with predictable cost, tenant isolation, and clear upgrade paths.

15.1 Deployment Targets & Environments

Goal.

Run the MVP with a managed-first posture: low ops, clear isolation, predictable costs, and straightforward scale-up paths. We separate *frontend*, *API*, *workers*, and *stateful services*.

15.1.1 Environment Matrix

dev Single-tenant sandbox; permissive CORS; all services may run locally except stateful managed backends.

staging Prod-like; same cloud regions and SKUs as prod; feature-flag validation; load/soak tests.

prod Managed DB/Storage/Redis; autoscaling API & workers; CDN in front of Next.js.

15.1.2 Cloud-Native Architecture

- **Frontend (Next.js, App Router):** Vercel (preferred) or Netlify.
  - *Why:* zero-config builds, CDN/edge caching, preview deployments.
  - *Config:* env vars (read-only), edge cache headers for static assets, API base URL per env.
- **API (NestJS) & Workers (BullMQ):** Fly.io or Render (simple) *or* AWS ECS/Fargate (advanced).
  - *Why (Fly/Render):* easy deploys, secrets store, autoscale; private networking to DB/Redis.
  - *Why (ECS/Fargate):* VPC control, IAM, SGs; future-proof for higher scale/compliance.
  - *NestJS layout:* monorepo with `apps/api` (HTTP) and `apps/worker` (queues). Each is a separate Nest application.
    - **API app:** `AppModule` + feature modules (`RfpsModule`, `OfferingsModule`, `AssistantModule`); `AuthGuard` derives `orgId` from JWT; a `PgScopeInterceptor` sets `SET LOCAL app.org_id=$1`.
    - **Worker app:** `BullModule.forRoot(...)` + processors (`ParseDocProcessor`, `EmbedChunksProcessor`, `ScoreMatchesProcessor`); each processor injects a scoped PG client.
    - **Shared libs:** `@app/db` (PG provider), `@app/scoring` (keyword/rules/final score), `@app/text` (normalize/chunk).
  - *Process model:* one service for API (stateless HTTP), one or more services for workers with separate concurrency; scale them independently.
  - *Config/DI:* use `@nestjs/config`; map env vars per env; inject `OpenAI` client via a provider (`OpenAiProvider`) to allow test stubs.
  - *RLS enforcement:* wrap request handlers with a transaction + `SET LOCAL app.org_id`; expose a per-request `PgClient` via `REQUEST` scope or a custom provider; workers do the same per job.
  - *Health/ops:* enable `@nestjs/terminus` health checks (DB, Redis, OpenAI ping); add `pino`/`nestjs-pino` for structured logs; propagate `x-request-id`.
- **Redis Caching (Managed — Redis Cloud recommended):**
  - *Why managed:* built-in HA/replication, backups, operations tooling, and `rediss://` endpoints for encrypted transit. [11, 9]
  - *Durability/policy:* enable AOF with `appendfsync=everysec` (throughput-friendly, $\leq$ 1s potential loss) and set eviction policy `noeviction` to avoid silent key loss for BullMQ metadata. [**?**, **?**]
  - *Security:* require AUTH; prefer TLS endpoints (`rediss://`); restrict access via provider network rules/VPC peering. [9, 11]
  - *Connection:* configure `REDIS_URL` in both API and Worker services; keep Redis on a private network if self-hosting on Render/Fly.

- **Database (Postgres + pgvector):** Supabase Postgres (managed).
  - *Why:* built-in `vector`, connection pooling, backups/PITR, SQL console.
  - *Config:* enforce SSL; RLS enabled; pooled URL for app traffic.
- **Object Storage:** Supabase Storage (S3-compatible).
  - *Pattern:* client direct upload via signed URLs; server receives `storageUrl` and enqueues parse.
- **LLM Provider:** OpenAI API.
  - *Config:* outbound egress allowlist; org-level key; per-env rate limits.

15.1.3 Network & Access Topology

- **Public:** Next.js (via CDN), API HTTPS endpoint.
- **Private:** API/Workers ↔ Postgres/Redis/Storage over private links or provider-private networks.
- **Ingress:** HTTPS only; WAF/Rate limit at edge (basic IP/QPS caps on API).
- **Egress:** Restrict to OpenAI endpoints and managed backends; block general internet egress from workers if possible.

15.1.4 Secrets & Config

- Store secrets in platform secret managers (Vercel/Render/Fly secrets or AWS SSM/Secrets Manager). Never bake into images.
- Separate per-environment keys; rotate regularly; least-privileged DB users per service (API vs worker if needed).
- For Postgres tenancy, set `SET LOCAL app.org_id=$1` per-request/transaction in the API; enforce RLS policies.

15.1.5 Provisioning (Setup · Test · Verification)

15.1.5.1 Frontend (Next.js on Vercel)

15.1.5.1.1 Setup.

1. **Provision on Vercel (Dashboard).**
   (a) Vercel Dashboard → **New Project** → **Import** our Git repo (branch `main` as Production).
   (b) Framework is auto-detected as **Next.js**. Keep the default `Install` command and `Build Command` unless our repo is custom (we override below).
   (c) **Project Settings** → **General**: set **Production Branch** = `main`.
   (d) **Project Settings** → **Functions**: set **Serverless Function Region** (e.g., `iad1` or closest to our API).

2. **Connect repo & env.** Create a Vercel project from our Next.js repo. Set env vars under *Settings* → *Environment Variables* for all environments:

```
# Vercel -> Project Settings -> Environment Variables
# Scope: Production / Preview / Development (set per-scope as needed)
NEXT_PUBLIC_API_BASE_URL=https://api.example.com
NODE_ENV=production
```

3. **(Optional) API proxy rewrites (separate API domain).** If the API is on another host, add a `vercel.json` rewrite so the frontend fetches `/api/*` locally:

Listing 98: vercel.json (rewrites + headers)

```
{
  "rewrites": [
    { "source": "/api/(.*)", "destination": "https://api.example.com/$1" }
  ],
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        { "key": "Cache-Control",
          "value": "public, max-age=60, s-maxage=600, stale-while-revalidate=300" }
      ]
    }
  ]
}
```

4. **(Optional) Next.js config for prod.** Pin basic production options in `next.config.js`:

Listing 99: next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  poweredByHeader: false,
  images: { domains: ["images.example.com"] },
  experimental: { forceSwcTransforms: true }
};
module.exports = nextConfig;
```

5. **Build & deploy (commands).** Ensure our `package.json` has standard scripts; Vercel uses these automatically:

Listing 100: package.json (scripts)

```
{
  "scripts": {
    "dev": "next dev",
```

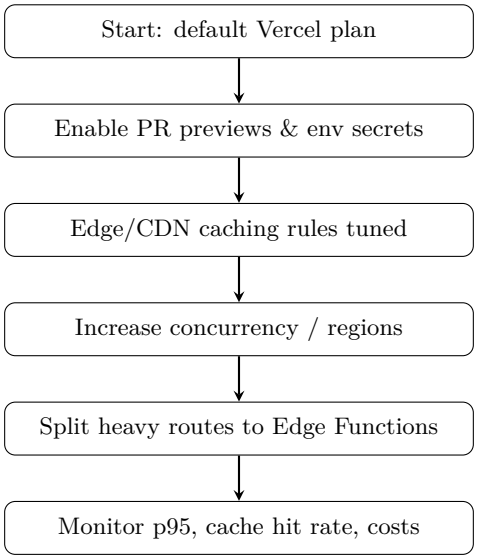
```
    "build": "next build",
    "start": "next start -p 3000",
    "lint": "next lint"
  }
}
```

In the project, leave **Build Command** as `npm run build` and **Output Directory** as `.next`. Enable **Preview Deployments** for PRs.

6. **Custom domain & TLS.** *Settings* → *Domains* → add `app.example.com`; create a `CNAME` to the Vercel target. Wait for `Verified`; TLS is auto-provisioned.
7. **(Optional) Protection.** *Settings* → *Security*: enable **Password Protection** for Preview; require **Git Branch Protection** for `main`.
8. **(Optional) Vercel CLI flow.** For local-to-prod deployments via CLI:

```
npm i -g vercel
vercel link                 # link local repo to Vercel project
vercel env pull .env.local  # pull envs for local dev
vercel --prod               # deploy current commit to Production
```

Autoscale at the edge/CDN; per-branch previews; project-scoped secrets. Start with default plan; scale via Vercel concurrency/regions as traffic grows.



15.1.5.1.2 Test.

1. **Smoke:** open the Vercel preview/production URL and confirm the app renders. Static assets must return `200` with `Cache-Control` from `vercel.json`:

```
curl -sI https://app.example.com/ | egrep 'HTTP/|cache-control|x-vercel-id'
```

2. **API wiring:** visit a page that calls the API (or `/api/health` if proxied) and verify `200` + correct base URL usage.
3. **Env:** render a page element that outputs `NEXT_PUBLIC_API_BASE_URL` to confirm it's injected for the current environment.
4. **Region check (functions):** hit a dynamic route and inspect response headers for `x-vercel-id` to confirm the intended functions region.

15.1.5.1.3 Results.

Area	Check / Command	Pass (Expected)
Build & deploy	Vercel deploy log	Status `READY`, no build errors.
Env injection	Page shows API base	Correct URL rendered.
Caching headers	`curl -I https://site.vercel.app/`	`Cache-Control` matches config.
Functions region	`x-vercel-id` header	Matches selected region (e.g., `iad1`).

15.1.5.2 API (NestJS on Render)

15.1.5.2.1 Setup.

1. **Provision on Render (Recommended).**
   (a) **Connect Git provider:** Render Dashboard → **New +** → **Blueprint from Repo** → pick the NestJS repo (branch `main`).
   (b) **Region & plan:** choose region (e.g., `Oregon`) and **Standard** plan for the web service.
   (c) **Create from `render.yaml`:** Render reads the `services:` entry (`name: api, type: web, env: docker`) and proposes resources.
   (d) **Env vars:** in the creation wizard, add `DATABASE_URL`, `REDIS_URL`, `NODE_ENV=production`. Leave `sync:false` vars to be supplied in the dashboard after creation if preferred.
   (e) **Health check:** confirm `/healthz` is set as `healthCheckPath` (from `render.yaml`). Keep **Auto Deploy** = `On push to main`.
   (f) **Create Blueprint:** click **Apply** to create the `api` service defined in `render.yaml`.
2. **(Alternative) Provision on Render (Console — manual).**
   (a) **New** → **Web Service** → **From Repo** → pick the repo/branch.
   (b) **Runtime:** set **Environment** = `Docker`. **Plan:** `Standard`. **Region:** select closest to users.
   (c) **Start command:** leave blank (Dockerfile `CMD` is used).
   (d) **Health check:** set `/healthz`. **Auto Deploy:** `Yes`.
   (e) **Env vars:** add `DATABASE_URL`, `REDIS_URL`, `NODE_ENV=production`. Save and **Create Web Service**.

3. **Post-deploy wiring (Render).**
   (a) **Custom domain:** *Settings* → *Custom Domains* → add `api.example.com`; create DNS `CNAME` to Render's hostname; wait for "`Verified`" and TLS auto-provisioning.
   (b) **Scaling:** *Settings* → *Scaling* → set **Instances** (start with 2) and enable autoscale if needed.
   (c) **Networking:** keep API public; keep internal services (workers/Redis) as **Private Services**.
   (d) **Secrets rotation:** rotate `DATABASE_URL`/`REDIS_URL` via *Environment* and redeploy.
   (e) **Observability:** *Logs* tab for live logs; *Shell* for debug; set alerts on error rate/latency in our APM or log pipeline.
4. **Dockerize API.** Add `Dockerfile`:

Listing 101: Dockerfile (NestJS API)

```
    FROM node:20-alpine AS deps
    WORKDIR /app
    COPY package*.json ./
    RUN npm ci --omit=dev

    FROM node:20-alpine AS build
    WORKDIR /app
    COPY . .
    RUN npm ci && npm run build

    FROM node:20-alpine
    WORKDIR /app
    ENV NODE_ENV=production
    COPY --from=build /app/dist dist
    COPY --from=deps /app/node_modules node_modules
    EXPOSE 10000
    CMD ["node", "dist/main.js"]

```

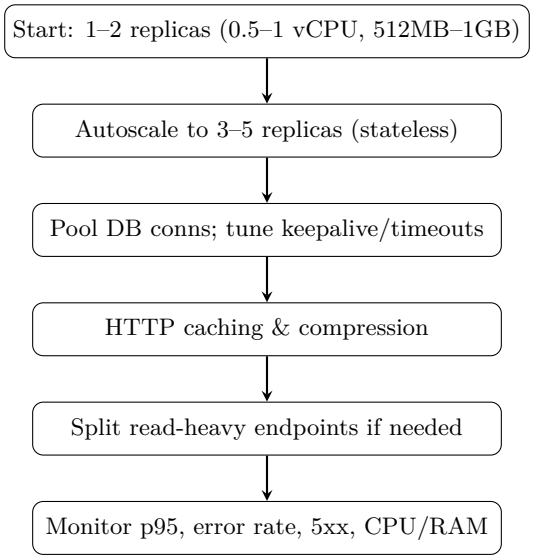
5. **Health endpoint.** Add `/healthz` in Nest (e.g., `@nestjs/terminus`) that returns `200`.
6. **Render service.** Add `render.yaml`:

Listing 102: render.yaml (API)

```
services:
  - name: api
    type: web
    env: docker
    plan: standard
    autoDeploy: true
    healthCheckPath: /healthz
```

```
    envVars:
      - key: NODE_ENV
        value: production
      - key: DATABASE_URL
        sync: false
      - key: REDIS_URL
        sync: false
```

Begin with *1–2* replicas (0.5–1 vCPU, 512MB–1GB RAM), autoscale to *3–5*. Stateless; horizontal scale first.



15.1.5.2.2 Test.

1. **Health:** `curl -sf https://api.onrender.com/healthz` → `200`.
2. **DB connectivity:** exercise an endpoint that hits Postgres and expects `200` + valid payload.
3. **Graceful shutdown:** trigger redeploy; observe no 5xx spikes (Nest handles `SIGTERM`).

15.1.5.2.3 Results.

Area	Check	Pass (Expected)
Health	`/healthz`	`200 OK`.
Postgres conn	API endpoint w/ DB	Query returns expected rows.
Redeploy	Render events/metrics	No 5xx spike; requests drain.

15.1.5.3 Workers (BullMQ on Render)

15.1.5.3.1 Setup.

Render provisioning for workers is the same flow as the API service we just change the service type and envs. Concretely: we use the same repo and `render.yaml` Blueprint, but define the worker as a `type: private_service` with a worker entrypoint, set `REDIS_URL`/`NODE_ENV` env vars, and keep networking private (no public URL). Scaling, deploys, logs, and secrets management follow the exact same procedures as the API setup.

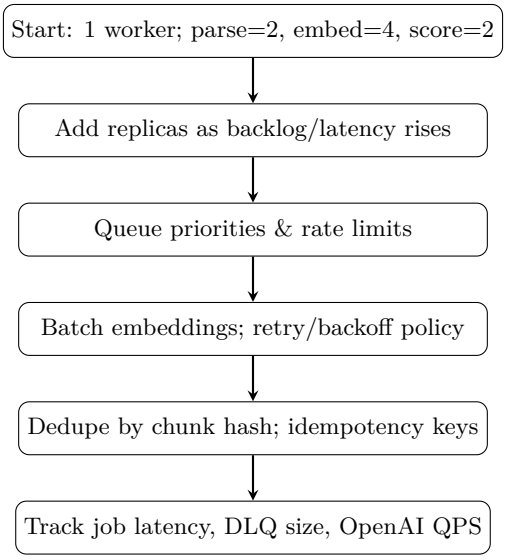
1. **Private Service.** Add to `render.yaml`:

Listing 103: render.yaml (Worker)

```
- name: worker
  type: private_service
  env: docker
  plan: standard
  autoDeploy: true
  envVars:
    - key: REDIS_URL
      sync: false
    - key: NODE_ENV
      value: production
```

2. **Worker entrypoint.** Start the BullMQ processors in `dist/worker/main.js`; ensure process exits non-zero on unrecoverable errors.

Start with *1* instance; tune per-queue concurrency (e.g., parse=2, embed=4, score=2). Scale by adding replicas and adjusting concurrency.



15.1.5.3.2 Test.

1. **Roundtrip:** enqueue a job via API; verify worker processes it and marks `completed`.

Listing 104: enqueue (example)

```
await scoreMatchesQ.add('smoke', { id: 'T1' });
```

2. **Metrics sanity:** check job counts (`wait/active/completed/failed`) through BullMQ APIs.

15.1.5.3.3 Results.

Area	Check	Pass (Expected)
Queue connectivity	Job → processed	State `completed`, result payload OK.
Failure path	Inject bad job	Appears in `failed` with error; retries per policy.

15.1.5.4 DB (PostgreSQL — Supabase Postgres + `pgvector`)

15.1.5.4.1 Setup.

1. **Create project & get production connection strings.** In Supabase Dashboard, create a project (choose region/compute). In *Project* → *Connect*, copy the **pooled connection string** (Supavisor/PgBouncer) for apps, and keep the **direct connection string** for migrations/long transactions.[23, 24]
2. **Enable pgvector (Dashboard).** *Database* → *Extensions* → search `vector` → **Enable**. This installs the `vector` extension in the project.[25, 26]
3. **Schema & indexing (SQL).** Apply the DDL below (dimension may vary by embedding model):

Listing 105: Enable pgvector & schema (IVFFLAT index)

```
-- Extension (no-op if already enabled by dashboard)
CREATE EXTENSION IF NOT EXISTS vector;

-- Sample table for chunks + embeddings (1536-d as example)
CREATE TABLE IF NOT EXISTS doc_chunks (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id UUID NOT NULL,
  content TEXT NOT NULL,
  embedding VECTOR(1536) NOT NULL
);

```

```
-- Approximate NN index (IVFFLAT) for L2 distance; tune lists for recall/speed
CREATE INDEX IF NOT EXISTS doc_chunks_embedding_ivfflat
  ON doc_chunks USING IVFFLAT (embedding VECTOR_L2_OPS) WITH (LISTS = 100);

ANALYZE doc_chunks;
```

Notes: pgvector supports **IVFFLAT** and **HNSW** approximate indexes; choose per workload and tune parameters (e.g., `lists` for IVFFLAT).[27, 28]

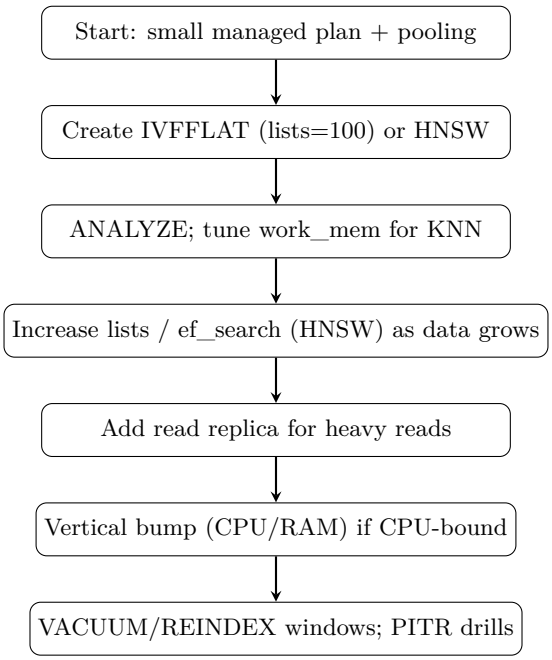
4. **Row Level Security (recommended).** Enable RLS and add tenant-aware policies:

Listing 106: RLS enable + tenant policy

```
ALTER TABLE doc_chunks ENABLE ROW LEVEL SECURITY;

CREATE POLICY "tenant can read own chunks"
ON doc_chunks FOR SELECT
USING (org_id = current_setting('app.org_id', true)::uuid);

CREATE POLICY "tenant can insert own chunks"
ON doc_chunks FOR INSERT
WITH CHECK (org_id = current_setting('app.org_id', true)::uuid);
```

In Supabase, tables created via the Table Editor have RLS on by default; for SQL-created tables we must enable it explicitly.[29]

Begin with a small managed plan; monitor CPU/IO/active conns. Increase `ivfflat.lists` or adopt `hnsw` as embeddings grow; enable connection pooling; add a read replica if reads surge.



15.1.5.4.2 Test.

1. **Connectivity (pooled).**

Listing 107: psql smoke (pooled URL)

```
psql "$DATABASE_URL" -c "SELECT 1;"
```

Expect `1`. Use the *pooled* URL for app paths.[23]

2. **Insert + KNN smoke.** Insert a row and run a nearest-neighbor query with the `<->` operator:

Listing 108: Insert + KNN query

```
INSERT INTO doc_chunks (org_id, content, embedding)
VALUES ('00000000-0000-0000-0000-000000000000', 'hello',
        '[0,0, ... ,0]'::vector);  -- replace with real embedding

SELECT id, content
FROM doc_chunks
ORDER BY embedding <-> '[0,0, ... ,0]'::vector
LIMIT 1;
```

The `<->` distance operator is provided by pgvector; IVFFLAT/HNSW speed up this ordering.[27, 30]

15.1.5.4.3 Results.

Area	Check	Pass (Expected)
Extension	`\dx`	`vector` present.
Index	`\di` on table	`ivfflat` (or `hnsw`) index exists.
KNN query	`ORDER BY <->`	Returns nearest row, no errors.
RLS	`pg_policies` view	Policies present; tenant read/insert constrained.
Conn. mode	App uses pooled URL	App endpoints succeed; long ops use direct URL.

15.1.5.5 Postgres storage/IOPS

15.1.5.5.1 Setup.

1. **Baseline (production provisioning).**
   (a) In Supabase *Project* → *Settings* → *Database*, pick a plan with sufficient disk and enable **Automated Backups**. For finer RPO, enable the **Point-in-Time Recovery (PITR)** add-on (Pro/Team/Enterprise), which allows restores to arbitrary timestamps within the retention window.[1]

(b) Understand that PITR relies on continuous **WAL** sequences; PostgreSQL recovery replays archived WAL to the target timestamp.[2]

2. **Maintenance windows (optional, large tables).**
   (a) Enable `pg_cron` in Supabase: *Database* → *Extensions* → enable `pg_cron`.[3]
   (b) Schedule **ANALYZE** (nightly) and **VACUUM** (weekly) for large or write-hot tables. PostgreSQL's autovacuum handles most cases, but explicit jobs give predictable windows.[6]

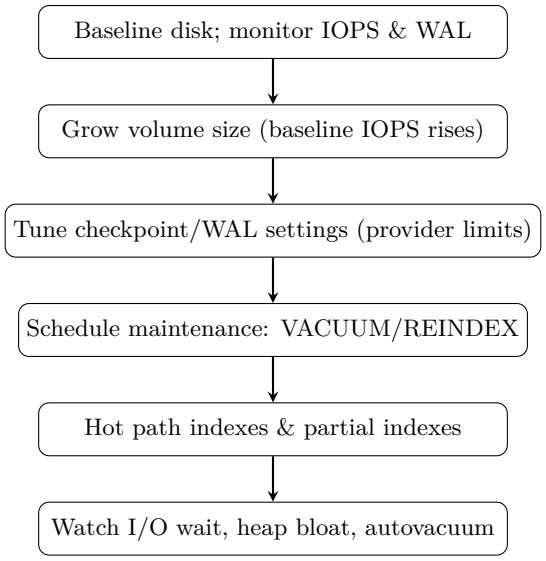
Listing 109: Enable pg_cron and schedule maintenance

```
-- Enable once (no-op if already enabled via Dashboard)
CREATE EXTENSION IF NOT EXISTS pg_cron;

-- Nightly ANALYZE at 02:00 UTC
SELECT cron.schedule('analyze-doc-chunks-nightly', '0 2 * * *',
  $$ANALYZE VERBOSE doc_chunks;$$);

-- Weekly VACUUM (non-FULL) at Sunday 03:00 UTC
SELECT cron.schedule('vacuum-doc-chunks-weekly', '0 3 * * 0',
  $$VACUUM VERBOSE doc_chunks;$$);
```

   (c) (Optional) Per-table autovacuum tuning for extreme write patterns; otherwise keep defaults.[6]

3. **IOPS/throughput considerations (cloud disks).**
   (a) Many cloud SSD volumes scale baseline **IOPS with volume size**; for example, AWS `gp2/gp3` scale linearly (e.g., ∼3 IOPS/GiB on `gp2`, with throughput limits that can cap effective IOPS for large I/O sizes).[31, 32]
   (b) Practically: provision adequate disk to raise baseline IOPS if needed; monitor I/O wait and WAL/heap activity before changing DB params.

Provision enough disk (baseline IOPS scales with size on many providers); watch WAL growth; schedule VACUUM/REINDEX windows; tune `work_mem`/`maintenance_work_mem` for index builds.



15.1.5.5.2 Test.

1. **Backups visible (Dashboard).** Confirm the most recent **automatic backup** exists; if PITR is enabled, confirm the **PITR slider/timestamp** is available for restore.[1]
2. **PITR drill (staging).** In a staging project, perform a *Restore to point in time* to a known timestamp and validate critical table row counts. Restoration time varies with WAL volume and time since last full backup.[33, 2]
3. **Maintenance jobs.** Verify cron jobs are registered and last runs succeeded:

Listing 110: Verify pg_cron jobs & last runs

```
SELECT jobid, jobname, schedule, active FROM cron.job ORDER BY jobid;
SELECT jobid, status, run_at FROM cron.job_run_details ORDER BY run_at DESC LIMIT
    10;
```

4. **Autovacuum/ANALYZE telemetry.** Inspect vacuum/analyze activity and table stats trend:

Listing 111: Observe vacuum/analyze activity

```
SELECT relname, n_tup_ins, n_tup_upd, n_tup_del,
       n_live_tup, n_dead_tup, last_vacuum, last_autovacuum,
       last_analyze, last_autoanalyze
FROM pg_stat_user_tables
ORDER BY n_dead_tup DESC
LIMIT 20;
```

15.1.5.5.3 Results.

Area	Check	Pass (Expected)
Backups	Console/backups list (PITR on/off)	At least 1 recent backup; PITR restore UI available if enabled.
Restore drill	Staging DB diff	Schema intact; row counts match snapshot/expectations.
Cron jobs	`cron.job`, `cron.job_run_details`	Jobs present; last status `SUCCEEDED`.
Vacuum/Analyze	`pg_stat_user_tables`	`last_autoanalyze/last_autovacuum` recent; dead tuples controlled.
IOPS headroom	Cloud metrics/APM	Low I/O wait; throughput within plan limits.

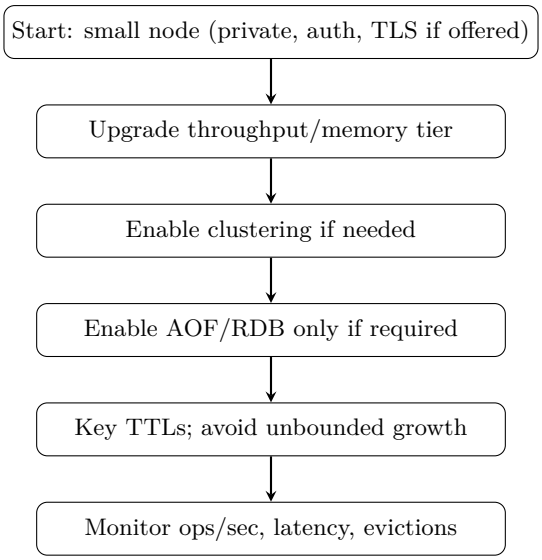
15.1.5.6 Redis (Managed — Redis Cloud recommended)

15.1.5.6.1 Setup.

1. **Create DB.** In Redis Cloud, create a database (e.g., `bullmq-prod`) in the target region (*Subscriptions* → *New Subscription*, choose cloud/region/plan).[11]
2. **Durability & safety.** Enable AOF with `everysec`; eviction policy `noeviction`; replication 1 primary + 1 replica; `rediss://` endpoint with AUTH.[**?**, **?**, 11, 9]
3. **Wire apps.** Set `REDIS_URL` in API/Worker (TLS via `rediss://` with password/auth):[9]

```
export REDIS_URL="rediss://default:pA$$-exAmplE@redis-xxxx.c12.us-east-1-1.ec2.redns
    .redis-cloud.com:12345"
```

Start with a small single node; keep private; enable persistence only if required; scale tier/throughput with load.



15.1.5.6.2 Test.

1. **TLS ping:** `redis-cli -u "$REDIS_URL" PING` → `PONG` (TLS negotiated with `rediss://`).[9]
2. **AOF enabled:** `redis-cli -u "$REDIS_URL" INFO persistence` contains `aof_enabled:1` and healthy rewrite fields.[**?**]

15.1.5.6.3 Results.

Area	Check	Pass (Expected)
Auth/TLS	`PING`	`PONG` with `rediss://`.
AOF policy	`CONFIG GET appendfsync`	`everysec`.
Eviction	`CONFIG GET maxmemory-policy`	`noeviction`.

15.1.5.7 Object Storage (Supabase Storage, S3-compatible)

15.1.5.7.1 Setup.

1. **Create bucket.** In the Supabase Dashboard, create a bucket (e.g., `uploads`) and choose visibility (public/private). Alternatively provision via API/CI:

Listing 112: Create bucket via JS API (private)

```
const { data, error } = await supabase.storage.createBucket('uploads', {
  public: false,                    // keep private in prod
  allowedMimeTypes: ['application/pdf','image/png','image/jpeg'],
  fileSizeLimit: 10485760           // 10 MB example
});
if (error) throw error;
```

   The create-bucket call and required permissions are documented in the Supabase Storage JS reference.[34]
2. **Access control (policies).** Configure bucket/object access with Storage access-control policies (backed by Postgres RLS on `storage.buckets`/`storage.objects`). For private buckets, we mint signed URLs from a trusted server and keep clients unprivileged.[37]
3. **Signed uploads (server route).** Implement a server endpoint that mints a *signed upload URL* for a specific object key; clients will `PUT` directly to Storage using that URL:

Listing 113: Mint signed upload URL (server)

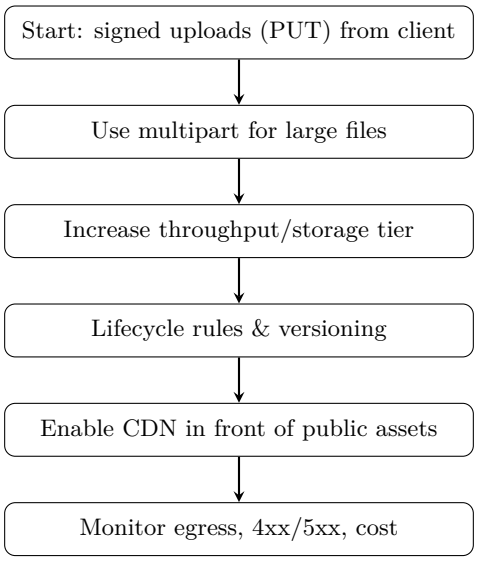
```
// Server-side (supabase-js with service role key)
const key = `org/${orgId}/${filename}`;
const { data, error } = await supabase.storage
  .from('uploads')
  .createSignedUploadUrl(key);      // returns signed URL + token
if (error) throw error;
```

```
return data;                          // { signedUrl, path, token }
```

The `createSignedUploadUrl` method and usage are documented in the Storage JS reference.[35]

4. **Client upload to signed URL.** The client (or CI) performs a direct `PUT` to the returned signed URL (no service key on the client). The official docs include the signed-upload flow and the companion "upload to signed URL" API.[36]
5. **(Optional) Public files.** For public assets, mark the bucket as public and/or generate public URLs for specific objects; see the Storage guide for public URL retrieval and caching guidance.[37]

Direct client uploads via signed URLs; scale via provider tier; use lifecycle rules for cost.



15.1.5.7.2 Test.

1. **PUT object (signed).** Use the signed URL to upload a small file; expect `200/204`. The "Upload to a signed URL" reference describes the exact request shape (headers/body).[36]
2. **List/get.** List the object path and GET the file (public) or GET via a short-lived signed URL (private), as described in the Storage guide.[37]

15.1.5.7.3 Results.

Area	Check	Pass (Expected)
Upload	PUT with signed URL	`200/204`; object appears in bucket.
Access	GET object	200 with correct bytes (per policy).

15.1.5.8 LLM (OpenAI API)

15.1.5.8.1 Setup.

1. **Create an API Platform account & org.** Sign in at the OpenAI API Platform and ensure an Organization exists for the project. The API uses *Bearer* authentication with secret keys managed under the Organization settings.[42]
2. **Create an API key (secret).** In the API Platform dashboard, open *API Keys* and select **Create new secret key**. Copy the key once (it cannot be viewed again) and store it in our secret manager.[42]
3. **(Optional) Scope/permissions.** For larger teams, create keys at the Project level and apply least-privilege permissions using the API Platform's key/role controls.[43]
4. **Harden access.** Enable **MFA** for the account/org to reduce key compromise risk.[44]
5. **Secrets.** Set `OPENAI_API_KEY` in API/Worker env (Render/Fly/ECS secret store). Example (Render `render.yaml`):

Listing 114: render.yaml (add OpenAI secret)

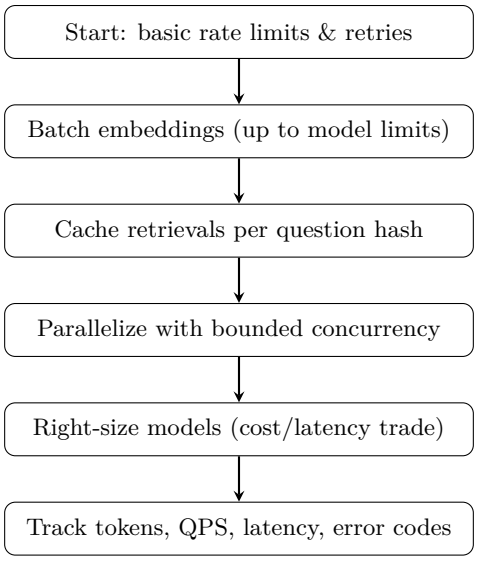
```
services:
  - name: api
    type: web
    env: docker
    envVars:
      - key: OPENAI_API_KEY
        sync: false   # set value in dashboard/CLI
```

6. **Client provider.** Inject the OpenAI client via a Nest provider so we can stub it in tests. Minimal example:

Listing 115: NestJS OpenAI provider

```
import OpenAI from "openai";
import { Module } from "@nestjs/common";

export const openaiFactory = () =>
  new OpenAI({ apiKey: process.env.OPENAI_API_KEY! });

@Module({
  providers: [{ provide: OpenAI, useFactory: openaiFactory }],
  exports: [OpenAI],
})
export class OpenAIModule {}
```

Enforce rate limits/backoff; batch embeddings; cache retrievals per question hash to reduce egress.



15.1.5.8.2 Test.

1. **Health (CLI).** Issue a lightweight embeddings request to validate auth; expect `200`:

Listing 116: curl smoke (embeddings)

```
curl https://api.openai.com/v1/embeddings \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"model":"text-embedding-3-small","input":"ping"}'
```

   (Authentication uses a Bearer token sourced from our API key.)[42]
2. **Health (app).** Call a tiny Nest endpoint that hits the OpenAI client once (e.g., small embedding) and assert `200`.
3. **Backoff.** Simulate 429 and verify retry/backoff logic triggers (e.g., exponential backoff with jitter).

15.1.5.8.3 Results.

Area	Check	Pass (Expected)
Auth	API call	`200`; quota shown or embedding returned.
Retry policy	429 simulation	Exponential backoff engaged; eventual success/fail logged.
Secrets	Config review	`OPENAI_API_KEY` present in secret store (not committed).

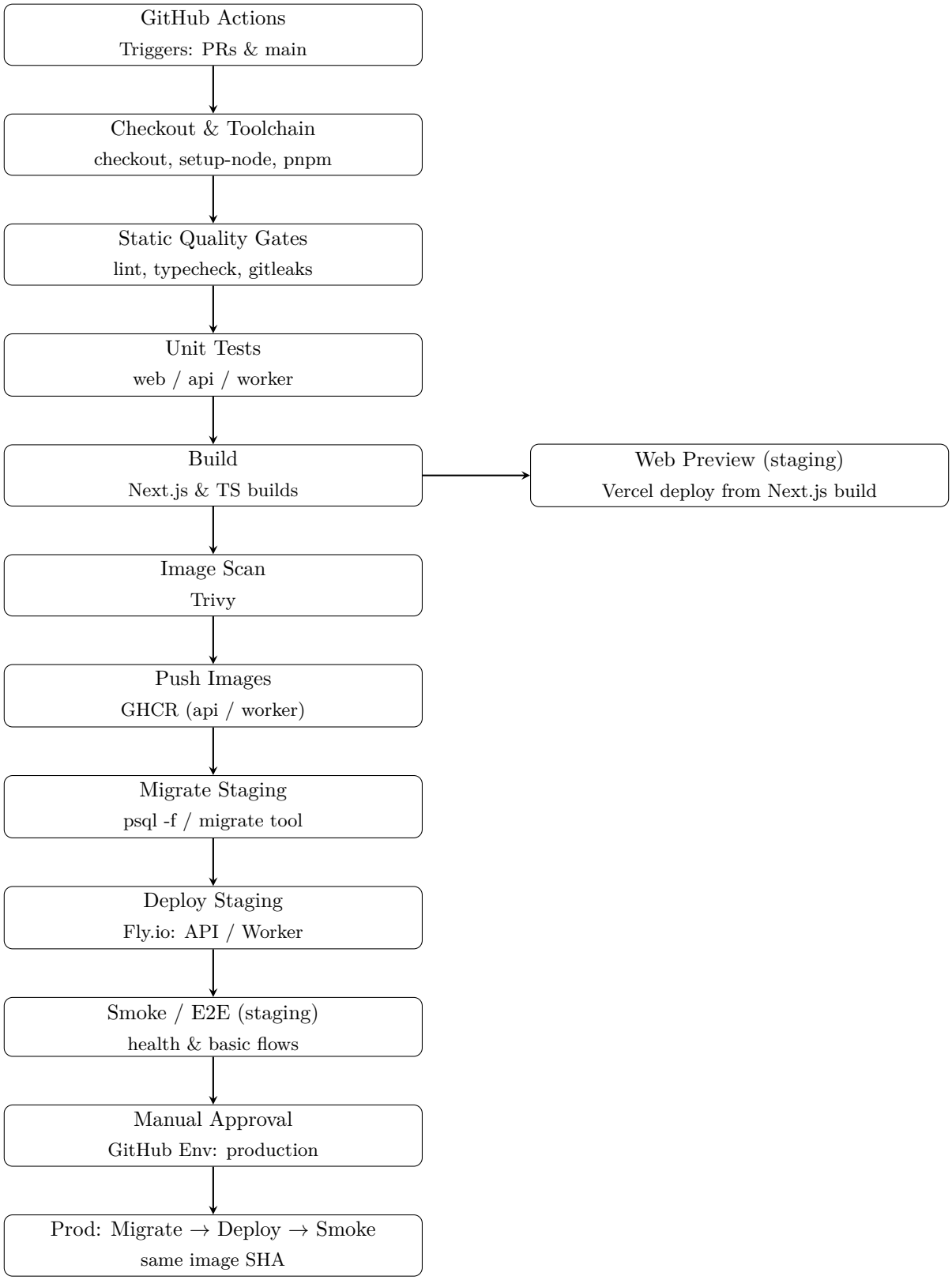
15.1.6 CI/CD Pipeline (GitHub Actions)

We standardize on **GitHub Actions** for CI/CD. The pipeline promotes the *same, immutable artifacts* from *staging* to *production* after manual approval. Web (Next.js) and backend (NestJS API & Worker) deploy independently to their best-fit platforms (Vercel; Fly/Render).

15.1.6.1 Workflows & triggers.

- **Web (Next.js)**: triggers on changes under `apps/web/**`.
- **Backend (API & Worker)**: triggers on changes under `apps/api/**`, `apps/worker/**`, `Dockerfile.{api,worker}`, `lib/**`.
- **Pull Requests**: run CI (lint, typecheck, unit tests, build) and publish *preview* deployments (Vercel previews).
- **Main branch**: on push, build & push images (API/Worker), migrate *staging*, deploy to *staging*, smoke/E2E test, *manual approval*, then migrate *prod* and deploy to *prod*.

15.1.6.2 Core Phases (per workflow).



Checkout & toolchain

- **What runs:** `actions/checkout@v4`, `actions/setup-node@v4` (Node 20 + pnpm cache), `pnpm/action-setup@v4`.
- **Purpose:** Recreate a clean, reproducible dev toolchain in CI.
- **Why it matters:** Deterministic Node version + a warmed pnpm cache keeps builds fast and eliminates "works on my machine" drift.
- **Intuitively:** This is similar to—Copy the recipe, pick the same oven (Node 20), and lay out pre-measured ingredients (pnpm cache) so baking is quick and consistent.

Static quality gates

- **What runs:** `pnpm -w lint` (ESLint), `pnpm -w typecheck` (TypeScript `-noEmit`), optional `gitleaks` scan.
- **Purpose:** Fail fast on cheap, objective problems (style, obvious bugs, unsafe patterns) before spending time on builds/tests.
- **Why it matters:** Highest-ROI checks—fast, non-flaky, and catch whole classes of issues pre-build.
- **Intuitively:** This is similar to—Run the spell-checker and grammar check before printing the book; it is cheap and saves us from reprinting later.

Unit tests

- **What runs:**
  - **web:** `pnpm -filter @app/web test` (Vitest/Jest)
  - **api/worker:** `pnpm -filter @app/api test`, `pnpm -filter @app/worker test`
- **Purpose:** Guard regressions in business logic, utilities, and adapters.
- **Why it matters:** Quick, stable feedback; high confidence without spinning full environments.
- **Intuitively:** This is similar to—Flip each light switch to make sure the room still lights up after rewiring.

Build

- **What runs:**
  - **web:** `pnpm -filter @app/web build` (Next.js)
  - **api/worker:** `pnpm -filter @app/api build`, `pnpm -filter @app/worker build`
  - **containers:** `docker/build-push-action@v6` for API/Worker images
- **Purpose:** Produce deployable artifacts: a Next.js build (for Vercel) and immutable Docker images (for Fly/Render).
- **Why it matters:** We promote the *same SHA* from staging → prod (no rebuilds), improving reproducibility and supply-chain hygiene.
- **Intuitively:** This is similar to—Bake the cake once, label it with a unique sticker (SHA), and move that exact cake from taste test to the party.

Artifact/image security (lightweight MVP)

- **What runs:** `aquasecurity/trivy-action` against built images.
- **Purpose:** Catch known CVEs and base-image issues before shipping.
- **Why it matters:** Low-effort net that blocks "we shipped a vulnerable libc" moments.
- **Intuitively:** This is similar to—Run a metal detector over our luggage before boarding.

Deploy

- **What runs:**
  - **web → Vercel:** `vercel-action` (or Vercel's native Git integration)
  - **api/worker → Fly.io:** `flyctl deploy` (staging first)
- **Purpose:** Put staging builds live with env-scoped secrets/URLs; validate end-to-end.
- **Why it matters:** Independent, best-fit platforms for web and backend; verify the exact artifacts we will promote.
- **Intuitively:** This is similar to—Put the prototype on a test shelf where people can actually touch it.

DB migrations (Supabase Postgres)

- **What runs:** `psql -f` (or `node-pg-migrate`/`Prisma migrate`), **staging first**, then manual approval, then **prod**.
- **Purpose:** Advance schema in lockstep with code; use **direct** DB URL for migrator, pooled URL for apps.
- **Why it matters:** Prevents runtime mismatches (code expecting columns that do not exist yet).
- **Intuitively:** This is similar to—Rearrange the furniture in the test room before moving the same plan into the live showroom.

Smoke/E2E checks

- **What runs:** `curl` health endpoints (smoke) and optionally `playwright` E2E against staging.
- **Purpose:** Verify the deploy is alive and basic flows work.
- **Why it matters:** Catches bad env vars, missing migrations, or routing issues right after deploy.
- **Intuitively:** This is similar to—Poke the app to see if it is breathing, then walk through the front door and one hallway.

Gates

- **What runs:** GitHub **Environment** protection for `production` (manual approval), promote **the same image SHA** from staging → prod.
- **Purpose:** Human checkpoint + immutable artifact promotion.
- **Why it matters:** Reduces risk and creates a clean audit trail.
- **Intuitively:** This is similar to—A final human thumbs-up before opening the doors, using the same cake we already tasted.

Workflow Summary

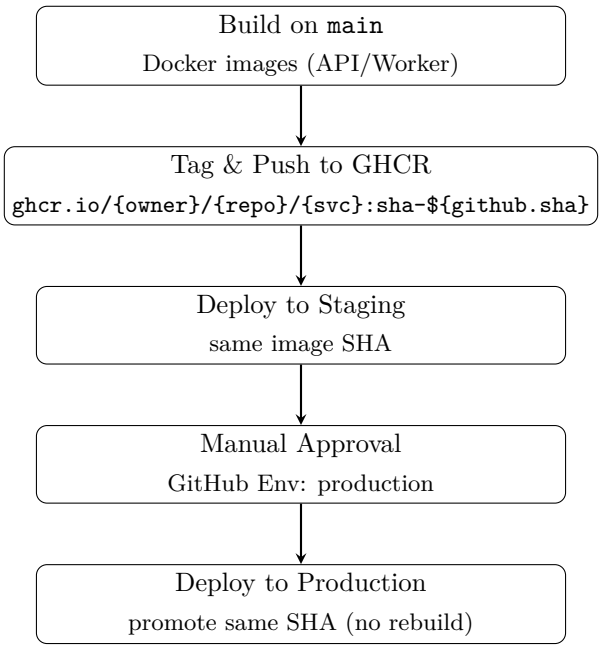
- **CI (on PR + main pushes):** Checkout/toolchain → Lint/Typecheck → Unit tests → Build → (optional) Image scan.
  Intuitively: Make sure the recipe is correct, ingredients are fresh, and the cake bakes.
- **CD (on main pushes):** Push images → **Staging** migrations → Deploy to staging (web+api/worker) → Smoke/E2E → **Manual approval** → **Prod** migrations → Deploy to prod (same SHA) → Smoke.
  Intuitively: Taste in the test kitchen, get a manager's nod, then serve the exact same cake at the party.

15.1.6.3 Environments & approvals.

- Define GitHub **Environments**: `staging`, `production`. Scope secrets per environment.
- Protect `production` with **required reviewers** (manual approval step).
- Use **environment URLs** for quick navigation to deployed dashboards/health pages.

15.1.6.4 Artifact promotion (immutability).

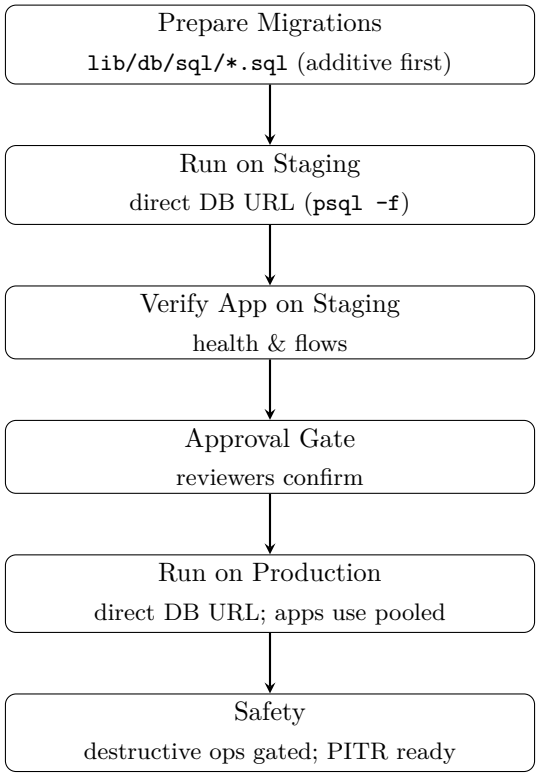
- Build Docker images once on `main` and push to `ghcr.io/{owner}/{repo}/{service}:sha-${github.sha}`.
- Deploy **the same image tags** to staging and, after approval, to production (no rebuilds).



Rationale: Ensures supply-chain integrity, reproducibility, and quick rollback; removes "works in staging, fails in prod" rebuild drift; gives a clear audit trail of what ran where.
Intuitively: Bake one cake, put a unique sticker on it (the SHA), and serve *that exact cake* first at tasting (staging) and then at the party (prod).

15.1.6.5 Database migrations (Supabase Postgres).

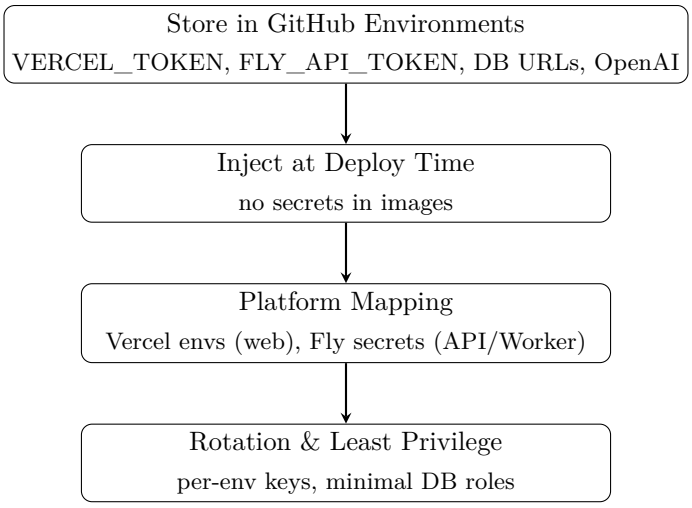
- **Order**: staging → approval → prod.
- **Connectivity**: use *direct* DB URL/role for migrations; apps use pooled URL.
- **Safety**: prefer backward-compatible migrations (additive first; destructive later) to avoid downtime.



Rationale: Prevents schema/code drift and minimizes outage risk; direct URLs avoid pool/RLS quirks during DDL; staging-first validates changes before customers see them.
Intuitively: The app has a shop key (can open the cash drawer and sell things). CI migrations have a contractor key (can move shelves and add a new counter). Nobody gets the building owner's master key.

15.1.6.6 Secrets & configuration.

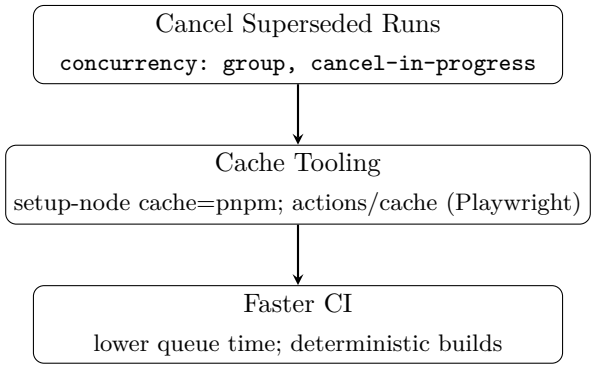
- Store CI/CD secrets in GitHub Environments (`VERCEL_TOKEN`, `FLY_API_TOKEN`, DB URLs, OpenAI keys).
- Never bake secrets into images; pass via platform secrets at deploy time (Vercel project envs; Fly secrets).



Rationale: Limits blast radius, enables rotation, and avoids leaking credentials through images, caches, or logs; separates build from runtime configuration.
Intuitively: Keep the keys in a safe and hand them to the chef *at service*, not sealed inside the oven.

15.1.6.7 Concurrency & caching.

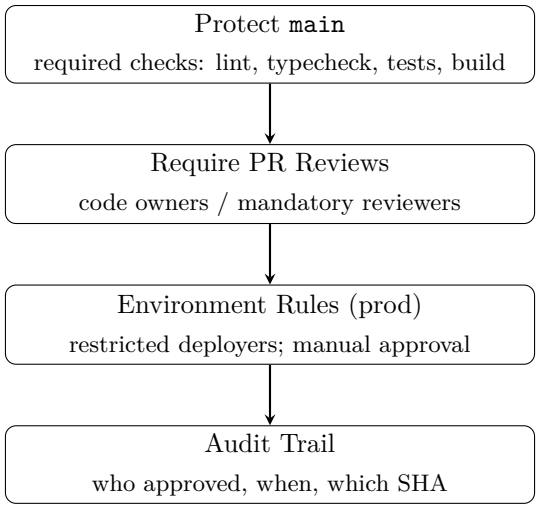
- Use `concurrency` groups to cancel superseded runs (e.g., multiple pushes to same branch).
- Enable `actions/cache` or Node cache in `setup-node` for `pnpm` and Playwright browsers (E2E).



Rationale: Prevents CI stampedes, reduces wait time and cost, and speeds repeatable steps while keeping builds deterministic.
Intuitively: Cancel old takeout orders when we place a new one; keep the pantry pre-stocked so cooking starts faster.

15.1.6.8 Branch protections & required checks.

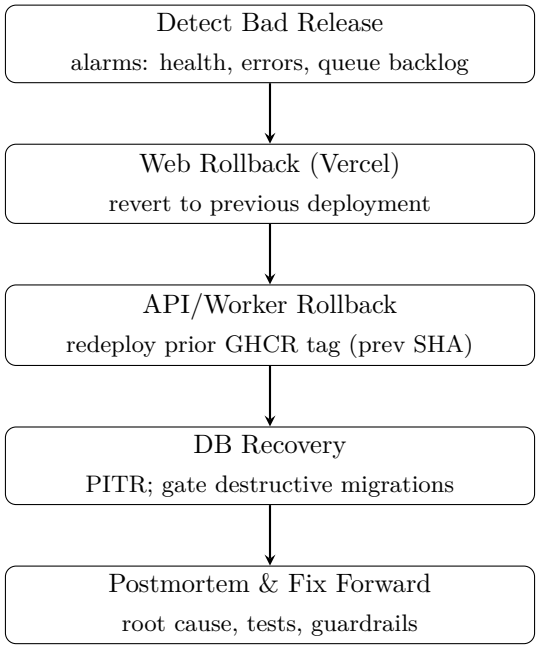
- Protect `main`: require status checks (lint, typecheck, tests, build) to pass before merge.
- Enforce PR reviews; restrict who can trigger production deployments via environment rules.



Rationale: Enforces quality gates and separation of duties; reduces accidental breakage; satisfies audit/compliance needs with clear approvals.
Intuitively: A bouncer checks tickets (status checks) and only managers hold the back-door key to the stage (prod deploy).

15.1.6.9 Rollback & recovery.

- **Web (Vercel)**: revert to previous deployment via Vercel dashboard/API.
- **API/Worker**: redeploy the last known-good `GHCR` image tag (e.g., previous SHA).
- **DB**: rely on managed PITR; keep destructive migrations gated behind separate approvals.

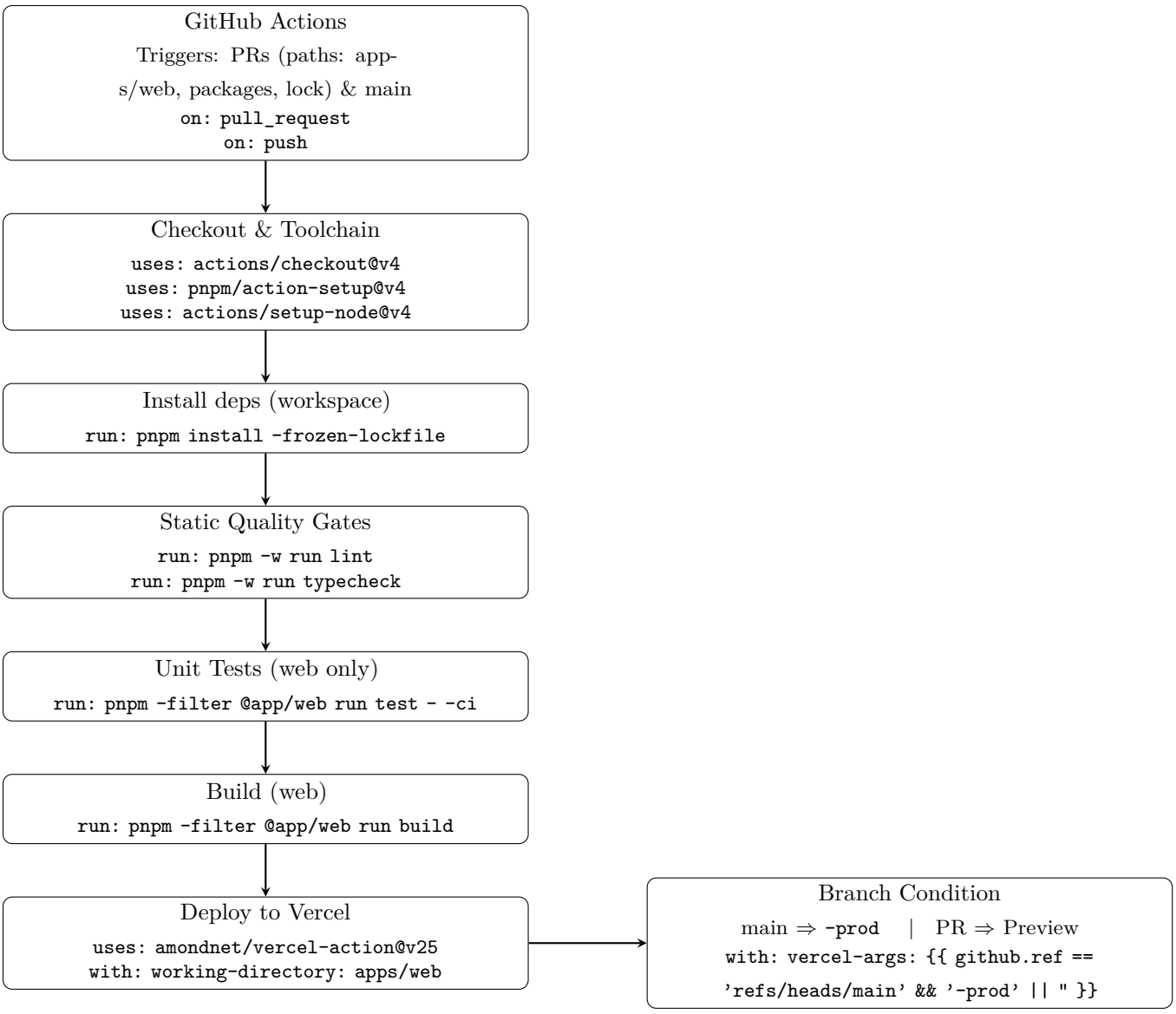


Rationale: Failures happen—fast, predictable rollback using immutable artifacts and provider features minimizes MTTR; PITR protects data; gating destructive DDL avoids irreversible mistakes.

Intuitively: Hit "undo" to the last save point; roll back the app to yesterday's build and the database to the moment before trouble.

15.1.6.10 Frontend (NextJS) GitHub Actions YAML sketch.

Triggers only when files under 'apps/web/' (or shared libs) change; deploys previews on PRs, prod on 'main'.



Listing 117: GitHub Actions (backend) — build once, stage, approve, prod

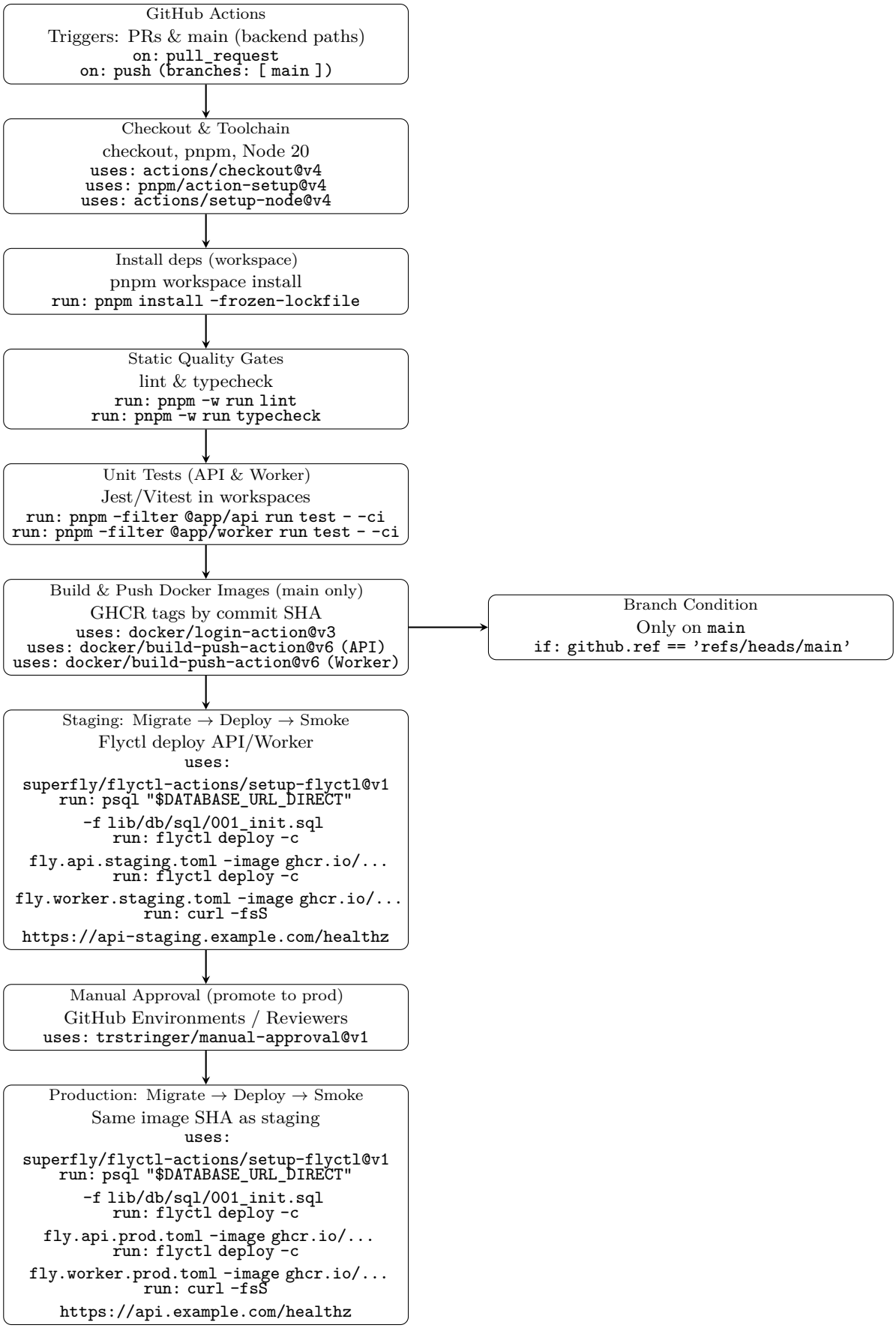
```
name: web (Next.js → Vercel)

on:
  pull_request:
```

```
    paths:
      - "apps/web/**"
      - "package.json"
      - "pnpm-lock.yaml"
      - "packages/**"
  push:
    branches: [ main ]
    paths:
      - "apps/web/**"
      - "package.json"
      - "pnpm-lock.yaml"
      - "packages/**"

jobs:
  build_and_deploy:
    runs-on: ubuntu-latest
    env:
      NODE_ENV: production
    steps:
      - uses: actions/checkout@v4

      - uses: pnpm/action-setup@v4
        with: { version: 9 }

      - name: Setup Node
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: "pnpm"

      - name: Install deps (workspace)
        run: pnpm install --frozen-lockfile

      - name: Lint & Typecheck
        run: |
          pnpm -w run lint
          pnpm -w run typecheck

      - name: Test (web only)
        run: pnpm --filter @app/web run test -- --ci

      - name: Build (web)
        run: pnpm --filter @app/web run build

```

```
      # Preview deploy on PR, Production on main
      - name: Deploy to Vercel
        uses: amondnet/vercel-action@v25
        with:
          vercel-token: ${{ secrets.VERCEL_TOKEN }}
          vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
          vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID_WEB }}
          working-directory: apps/web
          vercel-args: ${{ github.ref == 'refs/heads/main' && '--prod' || '' }}
```

15.1.6.11 Backend (NestJS/Node) GitHub Actions YAML sketch.

Triggers only when backend or shared libs change; builds and pushes Docker images, runs DB migrations (staging → prod with manual approval), then deploys.


GitHub Actions
Triggers: PRs & main (backend paths)
on: pull_request
on: push (branches: [main])
Checkout & Toolchain
checkout, pnpm, Node 20
uses: actions/checkout@v4
uses: pnpm/action-setup@v4
uses: actions/setup-node@v4
Install deps (workspace)
pnpm workspace install
run: pnpm install -frozen-lockfile
Static Quality Gates
lint & typecheck
run: pnpm -w run lint
run: pnpm -w run typecheck
Unit Tests (API & Worker)
Jest/Vitest in workspaces
run: pnpm -filter @app/api run test - -ci
run: pnpm -filter @app/worker run test - -ci
Build & Push Docker Images (main only)
GHCR tags by commit SHA
uses: docker/login-action@v3
uses: docker/build-push-action@v6 (API)
uses: docker/build-push-action@v6 (Worker)
Branch Condition
Only on main
if: github.ref == 'refs/heads/main'
Staging: Migrate → Deploy → Smoke
Flyctl deploy API/Worker
uses:
superfly/flyctl-actions/setup-flyctl@v1
run: psql "$DATABASE_URL_DIRECT"
-f lib/db/sql/001_init.sql
run: flyctl deploy -c
fly.api.staging.toml -image ghcr.io/...
run: flyctl deploy -c
fly.worker.staging.toml -image ghcr.io/...
run: curl -fsS
https://api-staging.example.com/healthz
Manual Approval (promote to prod)
GitHub Environments / Reviewers
uses: trstringer/manual-approval@v1
Production: Migrate → Deploy → Smoke
Same image SHA as staging
uses:
superfly/flyctl-actions/setup-flyctl@v1
run: psql "$DATABASE_URL_DIRECT"
-f lib/db/sql/001_init.sql
run: flyctl deploy -c
fly.api.prod.toml -image ghcr.io/...
run: flyctl deploy -c
fly.worker.prod.toml -image ghcr.io/...
run: curl -fsS
https://api.example.com/healthz

Listing 118: GitHub Actions (backend) — build once, stage, approve, prod

```yaml
name: backend (NestJS → Fly)

on:
  pull_request:
    paths: ["apps/api/**","apps/worker/**","Dockerfile.*","lib/**","package.json","pnpm-lock.yaml"]
  push:
    branches: [ main ]
    paths: ["apps/api/**","apps/worker/**","Dockerfile.*","lib/**","package.json","pnpm-lock.yaml"]

jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: pnpm/action-setup@v4
        with: { version: 9 }
      - uses: actions/setup-node@v4
        with: { node-version: 20, cache: pnpm }
      - run: pnpm install --frozen-lockfile
      - run: pnpm -w run lint
      - run: pnpm -w run typecheck
      - run: pnpm --filter @app/api run test -- --ci
      - run: pnpm --filter @app/worker run test -- --ci

  build_and_push_images:
    needs: ci
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - uses: docker/build-push-action@v6
        with:
          context: .
          file: Dockerfile.api
          push: true
          tags: ghcr.io/${{ github.repository }}/api:sha-${{ github.sha }}
```

```
      - uses: docker/build-push-action@v6
        with:
          context: .
          file: Dockerfile.worker
          push: true
          tags: ghcr.io/${{ github.repository }}/worker:sha-${{ github.sha }}

  stage_migrate_and_deploy:
    needs: build_and_push_images
    runs-on: ubuntu-latest
    environment: staging
    steps:
      - uses: actions/checkout@v4
      - uses: superfly/flyctl-actions/setup-flyctl@v1
      - name: DB migrate (staging)
        env: { DATABASE_URL_DIRECT: ${{ secrets.DATABASE_URL_DIRECT_STAGING }} }
        run: psql "$DATABASE_URL_DIRECT" -f lib/db/sql/001_init.sql
      - name: Deploy API (staging)
        run: flyctl deploy -c fly.api.staging.toml --image ghcr.io/${{ github.repository
    }}/api:sha-${{ github.sha }} --detach
      - name: Deploy Worker (staging)
        run: flyctl deploy -c fly.worker.staging.toml --image ghcr.io/${{ github.
    repository }}/worker:sha-${{ github.sha }} --detach
      - name: Smoke
        run: curl -fsS https://api-staging.example.com/healthz

  approve_and_promote_prod:
    if: github.ref == 'refs/heads/main'
    needs: stage_migrate_and_deploy
    runs-on: ubuntu-latest
    environment:
      name: production
    steps:
      - name: Await manual approval
        uses: trstringer/manual-approval@v1
        with:
          repo-token: ${{ secrets.GITHUB_TOKEN }}
          approvers: ${{ secrets.RELEASE_APPROVERS }}
          minimum-approvals: 1
          issue-title: "Promote backend to production"
          issue-body: "Approve SHA ${{ github.sha }}"

  prod_migrate_and_deploy:
    needs: approve_and_promote_prod
```

```
    runs-on: ubuntu-latest
    environment: production
    steps:
      - uses: actions/checkout@v4
      - uses: superfly/flyctl-actions/setup-flyctl@v1
      - name: DB migrate (prod)
        env: { DATABASE_URL_DIRECT: ${{ secrets.DATABASE_URL_DIRECT_PROD }} }
        run: psql "$DATABASE_URL_DIRECT" -f lib/db/sql/001_init.sql
      - name: Deploy API (prod)
        run: flyctl deploy -c fly.api.prod.toml --image ghcr.io/${{ github.repository }}/
    api:sha-${{ github.sha }} --detach
      - name: Deploy Worker (prod)
        run: flyctl deploy -c fly.worker.prod.toml --image ghcr.io/${{ github.repository
    }}/worker:sha-${{ github.sha }} --detach
      - name: Smoke
        run: curl -fsS https://api.example.com/healthz
```

15.1.6.12 Rationale.

- **Fast feedback**: cheap gates first (lint/typecheck/tests), then builds.
- **Safety**: staging-first DB migrations and deploy, smoke checks, then human approval.
- **Simplicity**: Actions-only; no extra CI servers. Real-world deploys to Vercel (web) and Fly/Render (backend).
- **Reproducibility**: promote the exact image SHA; no "works in staging but rebuilt for prod" drift.

15.1.7 Observability & Ops

15.1.7.1 Tooling options.

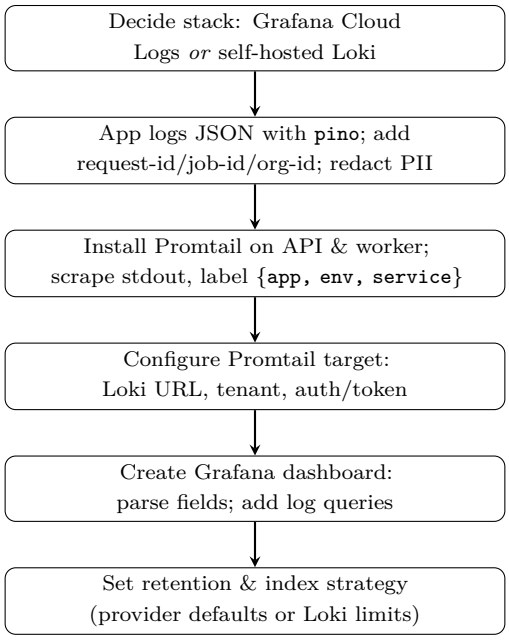
- **Logging**
  - *Cloud-native:* Grafana Cloud Logs (Loki), Datadog Logs, New Relic Logs.
  - *Self-hosted:* Loki + Promtail + Grafana. App logs via `pino` (JSON) with `request-id`, `job-id`, `org-id`; redact PII at source.
- **Metrics**
  - *Cloud-native:* Grafana Cloud Metrics (Prometheus remote-write), Datadog Metrics.
  - *Self-hosted:* Prometheus + Grafana; export with `prom-client` (Node), Postgres exporter for DB, custom queue metrics.
- **Tracing**
  - *Cloud-native:* Grafana Tempo Cloud, Sentry Performance, Datadog APM.
  - *Self-hosted:* OpenTelemetry SDK (Node) → OTel Collector → Tempo (+ Grafana). Propagate `traceparent` across API ↔ worker.

- **Alerting / On-call**
  - *Cloud-native:* Grafana Cloud Alerting, Datadog Monitors, Sentry Alerts, PagerDuty.
  - *Self-hosted:* Alertmanager (Prometheus) → Slack/PagerDuty. SLOs: API p95, queue latency, OpenAI error rate, DB KNN durations.

15.1.7.2 Logging (Loki stack)

This self-hosted stack is great for dev/staging. In production we can swap Loki/Grafana for managed (Grafana Cloud Logs) with Promtail agents. Works with Node (NestJS API + BullMQ workers) printing JSON via pino

`Setup Workflow`



Listing 119: docker-compose.yml (Loki + Promtail + Grafana)

```
version: "3.8"

services:
  loki:
    image: grafana/loki:2.9.4
    command: [ "-config.file=/etc/loki/config.yml" ]
    ports: [ "3100:3100" ]
    volumes:
      - ./loki:/etc/loki
      - loki-data:/loki
    restart: unless-stopped

  promtail:
    image: grafana/promtail:2.9.4
```

```
    command: [ "-config.file=/etc/promtail/config.yml" ]
    volumes:
      - ./promtail:/etc/promtail
      - /var/log:/var/log:ro
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on: [ loki ]
    restart: unless-stopped

  grafana:
    image: grafana/grafana:10.4.5
    ports: [ "3000:3000" ]
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-data:/var/lib/grafana
    depends_on: [ loki ]
    restart: unless-stopped

volumes:
  loki-data:
  grafana-data:
```

Loki configuration (`./loki/config.yml`).

Single-binary with *boltdb-shipper* and local filesystem storage. Seven-day retention for dev/staging. In production, prefer object storage with compactor + retention policies, or a managed Loki.

Listing 120: Loki config (boltdb-shipper, local FS, 7d retention)

```
auth_enabled: false  # put behind reverse-proxy if exposed

server:
  http_listen_port: 3100

common:
  instance_addr: 127.0.0.1
  path_prefix: /loki
  storage:
    filesystem:
      chunks_directory: /loki/chunks
      rules_directory: /loki/rules
  replication_factor: 1
  ring:
```

```
    kvstore:
      store: inmemory

schema_config:
  configs:
    - from: 2024-01-01
      store: boltdb-shipper
      object_store: filesystem
      schema: v13
      index:
        prefix: index_
        period: 24h

compactor:
  working_directory: /loki/compactor
  compactor_ring:
    kvstore:
      store: inmemory
  retention_enabled: true

limits_config:
  reject_old_samples: true
  reject_old_samples_max_age: 168h  # 7 days
  retention_period: 168h

ruler:
  alertmanager_url: http://localhost:9093

analytics:
  reporting_enabled: false
```

Promtail configuration (`./promtail/config.yml`).

Scrapes Docker container logs and system logs; forwards to Loki. Labels align with our services (e.g., `service=api|worker|web`).

Listing 121: Promtail config (Docker + system logs, safe labels)

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
```

```
  - url: http://loki:3100/loki/api/v1/push
    # basic_auth:
    #   username: loki
    #   password: secret

scrape_configs:
  # --- Docker containers (JSON) ---
  - job_name: docker
    docker_sd_configs:
      - host: unix:///var/run/docker.sock
        refresh_interval: 5s
    pipeline_stages:
      - json:
          expressions:
            log: log
            stream: stream
            time: time
          timestamp:
            source: time
            format: RFC3339Nano
            fallback_formats: [ RFC3339 ]
      - labels:
          image: __meta_docker_image
      - output:
          source: log
    relabel_configs:
      - source_labels: [__meta_docker_container_name]
        target_label: container
        regex: "/(.*)"
      - source_labels: [__meta_docker_container_log_stream]
        target_label: stream
      - source_labels: [__meta_docker_container_label_com_docker_compose_service]
        target_label: service
      - target_label: app
        replacement: api  # change per host (api/worker/web)

  # --- System logs example ---
  - job_name: syslog
    static_configs:
      - targets: [ localhost ]
        labels:
          job: syslog
          __path__: /var/log/*.log
```

Node app logging (JSON with `pino` + request-id/org-id).

Attach `pino-http`; generate/propagate `x-request-id`; attach `orgId` (from JWT guard in the real API). In our architecture, API & worker *print JSON to stdout*, which Promtail scrapes.

Listing 122: pino-http setup with redaction & request-id

```
import pinoHttp from 'pino-http';
import { randomUUID } from 'crypto';

export const httpLogger = pinoHttp({
  redact: { paths: ['req.headers.authorization', 'user.password'], censor: '[REDACTED]' },
  genReqId: (req) => (req.headers['x-request-id'] as string) || randomUUID(),
  customProps: (req) => ({
    service: 'api',
    env: process.env.NODE_ENV || 'dev',
    orgId: (req as any).orgId ?? 'unknown',
  }),
  transport: process.env.NODE_ENV === 'development'
    ? { target: 'pino-pretty', options: { colorize: true } }
    : undefined
});
```

Listing 123: Express wiring (derive orgId; emit health)

```
// server.ts
import express from 'express';
import { httpLogger } from './logger';

const app = express();

app.use((req, _res, next) => {
  // In real app, derive from JWT (AuthGuard) and set req.user/orgId.
  (req as any).orgId = req.header('x-org-id') || 'unknown';
  next();
});

app.use(httpLogger);

app.get('/health', (_req, res) => res.json({ ok: true }));

app.listen(3000, () => console.log('API on :3000'));
```

Grafana → Loki wiring.

1. Open Grafana at `http://localhost:3000` (admin/admin).

2. **Add Data Source** → Loki → URL: `http://loki:3100` → *Save & Test*.
3. Create a dashboard and add a *Logs* panel (query examples below).

LogQL examples.

Listing 124: Recent logs for API service

```
{service="api"} | json | line_format "{{.message}}"
```

Listing 125: Filter by orgId and requestId

```
{service="api"} | json | orgId="acme-co" | requestId="d7c4..."
```

Listing 126: Error count by container over 5m

```
sum by (container) (count_over_time({service="api"} |= "error" [5m]))
```

Listing 127: Latency histograms if we log durationMs

```
sum by (route) (rate(({service="api"} | json | unwrap durationMs)[5m]))
```

Security & multi-tenancy notes.

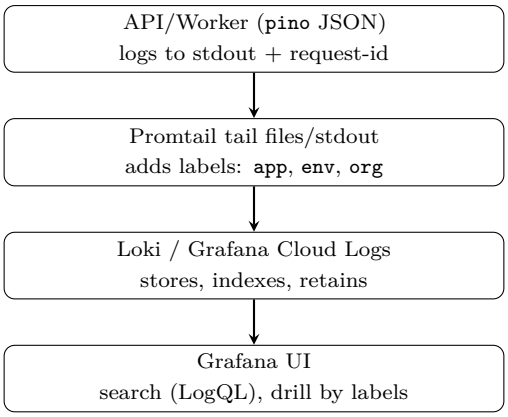
- Put Loki behind TLS and a reverse proxy (nginx/traefik). If multi-tenant, enable `auth_enabled: true` and use `X-Scope-OrgID`.
- Avoid high-cardinality labels (e.g., raw user IDs). Parse as fields with `| json` and filter at query time instead.

Quick push (without Promtail).

Listing 128: POST a test log to Loki directly

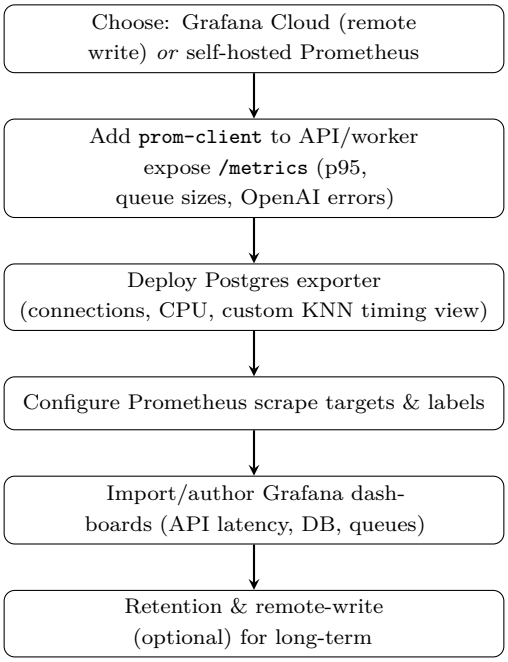
```
curl -X POST "http://localhost:3100/loki/api/v1/push" \
  -H "Content-Type: application/json" \
  --data-raw '{
    "streams": [{
      "stream": { "app":"api", "env":"dev" },
      "values": [[ "'$(date +%s%N)'", "{\"level\":\"info\",\"msg\":\"hello from curl\"}"
    ]]
    }]
  }'
```

Logging — runtime data flow.



15.1.7.3 Metrics (Prometheus)

Setup Workflow



This configuration provides a production-lean, self-hosted Prometheus stack aligned with our architecture (NestJS API, BullMQ workers, Supabase PostgreSQL, optional cAdvisor/node-exporter). Services expose `/metrics` via `prom-client`; Prometheus scrapes them; Grafana visualizes.

docker-compose.yml — Prometheus, Grafana, exporters.

Listing 129: docker-compose.yml

```
version: "3.9"

services:
  prometheus:
    image: prom/prometheus:v2.53.0
```

```
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --web.enable-lifecycle
    ports: ["9090:9090"]
    volumes:
      - ./prometheus:/etc/prometheus
      - prom-data:/prometheus
    restart: unless-stopped

  grafana:
    image: grafana/grafana:10.4.6
    ports: ["3000:3000"]
    environment:
      GF_SECURITY_ADMIN_USER: admin
      GF_SECURITY_ADMIN_PASSWORD: admin
    volumes:
      - graf-data:/var/lib/grafana
    restart: unless-stopped
    depends_on: [prometheus]

  # PostgreSQL exporter (targets Supabase/Postgres)
  postgres_exporter:
    image: prometheuscommunity/postgres-exporter:v0.15.0
    environment:
      DATA_SOURCE_NAME: ${PG_EXPORTER_DSN}
      # PG_EXPORTER_EXTEND_QUERY_PATH: /etc/queries/queries.yaml
    volumes:
      - ./postgres_exporter:/etc/queries:ro
    restart: unless-stopped

  # Optional: cAdvisor (container metrics on a single host)
  cadvisor:
    image: gcr.io/cadvisor/cadvisor:v0.47.2
    ports: ["8080:8080"]
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:ro
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    restart: unless-stopped

  # Optional: Node exporter for host metrics
  node_exporter:
    image: prom/node-exporter:v1.8.2
```

```
    pid: host
    network_mode: host
    restart: unless-stopped

volumes:
  prom-data:
  graf-data:
```

Env note. Set a read-only DSN for the exporter:
`export PG_EXPORTER_DSN="postgres://user:password@host:6543/postgres?sslmode=require"`.

prometheus.yml — scrape configs, labels, rules.

Listing 130: prometheus/prometheus.yml

```
global:
  scrape_interval: 15s
  evaluation_interval: 30s
  external_labels:
    cluster: "prod"

rule_files:
  - rules.yml

scrape_configs:
  # --- API (NestJS) ---
  - job_name: "api"
    metrics_path: /metrics
    static_configs:
      - targets: ["api.internal:3001"]   # private addr or public URL
        labels:
          service: api
          env: prod

  # --- Worker (BullMQ) ---
  - job_name: "worker"
    metrics_path: /metrics
    static_configs:
      - targets: ["worker.internal:3002"]
        labels:
          service: worker
          env: prod

  # --- Postgres exporter ---
  - job_name: "postgres"
    static_configs:
```

```
      - targets: ["postgres_exporter:9187"]
        labels:
          service: postgres
          env: prod

  # --- cAdvisor (optional) ---
  - job_name: "cadvisor"
    static_configs:
      - targets: ["cadvisor:8080"]
        labels:
          service: cadvisor
          env: prod

  # --- Node exporter (optional) ---
  - job_name: "node"
    static_configs:
      - targets: ["localhost:9100"]
        labels:
          service: node
          env: prod
```

Listing 131: prometheus/rules.yml — recording & alert rules

```
groups:
- name: recording
  interval: 30s
  rules:
    # API p95 latency from Histogram over 5m
    - record: job:http_p95_ms:5m
      expr: |
        histogram_quantile(
          0.95,
          sum by (le) (rate(http_request_duration_ms_bucket{service="api"}[5m]))
        ) * 1.0

    # Worker queue size (example Gauge)
    - record: job:queue_waiting:sum
      expr: sum(worker_queue_waiting)

- name: alerts
  rules:
    - alert: ApiHighErrorRate
      expr: sum(rate(http_requests_total{service="api",code=~"5.."}[5m])) /
            sum(rate(http_requests_total{service="api"}[5m])) > 0.05
      for: 10m
```

```
      labels: { severity: page }
      annotations:
        summary: "API 5xx error rate > 5%"

    - alert: PostgresConnectionsHigh
      expr: pg_stat_activity_count{datname=~".+"} > 0.8 * pg_settings_max_connections
      for: 10m
      labels: { severity: warn }
      annotations:
        summary: "Postgres nearing max connections"
```

API /metrics (NestJS/Express) via prom-client.

Listing 132: apps/api/src/metrics.ts

```
import client from 'prom-client';
import type { Request, Response } from 'express';

client.collectDefaultMetrics({ prefix: 'api_' });

export const httpReqCounter = new client.Counter({
  name: 'http_requests_total',
  help: 'HTTP request count',
  labelNames: ['method','route','code'],
});

export const httpDuration = new client.Histogram({
  name: 'http_request_duration_ms',
  help: 'HTTP request duration (ms)',
  buckets: [5,10,25,50,100,250,500,1000,2500,5000],
  labelNames: ['method','route','code'],
});

export const openaiErrors = new client.Counter({
  name: 'openai_api_errors_total',
  help: 'OpenAI API error count',
  labelNames: ['operation','status'],
});

export function metricsHandler(_req: Request, res: Response) {
  res.set('Content-Type', client.register.contentType);
  res.end(client.register.metrics());
}

// Express/Nest adapter example (Express)
export function promMiddleware(routeKey: (req: Request)=>string) {
```

```
  return async (req: Request, res: Response, next: Function) => {
    const end = httpDuration.startTimer({ method: req.method, route: routeKey(req) });
    res.on('finish', () => {
      httpReqCounter.inc({ method: req.method, route: routeKey(req), code: String(res.
    statusCode) });
      end({ code: String(res.statusCode) });
    });
    next();
  };
}
```

Listing 133: apps/api/src/server.ts — wiring /metrics

```
import express from 'express';
import { metricsHandler, promMiddleware } from './metrics';
import { openaiErrors } from './metrics';

const app = express();

// Minimal route keyer (avoid exploding cardinality)
const routeKey = (req: any) => (req.route?.path || req.path || 'unknown');

// Apply before our routes
app.use(promMiddleware(routeKey));

app.get('/metrics', metricsHandler);

// Example: count OpenAI failures
async function callOpenAI() {
  try {
    // ... OpenAI call ...
  } catch (e: any) {
    openaiErrors.inc({ operation: 'embeddings', status: String(e?.status || 500) });
    throw e;
  }
}

app.listen(3001, () => console.log('API on :3001'));
```

Worker metrics (BullMQ queue sizes).

Listing 134: apps/worker/src/metrics.ts

```
import client from 'prom-client';
import { Queue } from 'bullmq';

```

```
client.collectDefaultMetrics({ prefix: 'worker_' });

export const queueWaiting = new client.Gauge({
  name: 'worker_queue_waiting',
  help: 'Number of waiting jobs',
  labelNames: ['queue'],
});
export const queueActive = new client.Gauge({
  name: 'worker_queue_active',
  help: 'Number of active jobs',
  labelNames: ['queue'],
});

export async function pollQueues(queues: Record<string, Queue>) {
  for (const [name, q] of Object.entries(queues)) {
    const [w, a] = await Promise.all([q.getWaitingCount(), q.getActiveCount()]);
    queueWaiting.set({ queue: name }, w);
    queueActive.set({ queue: name }, a);
  }
}
```

Listing 135: apps/worker/src/main.ts — expose /metrics

```
import http from 'http';
import client from 'prom-client';
import { queues } from './queue';
import { pollQueues } from './metrics';

// Simple /metrics server
const srv = http.createServer(async (req, res) => {
  if (req.url === '/metrics') {
    await pollQueues({ parse: queues.parseDoc.q, embed: queues.embedChunks.q, score:
    queues.scoreMatches.q });
    res.writeHead(200, { 'Content-Type': client.register.contentType });
    res.end(await client.register.metrics());
  } else {
    res.writeHead(404); res.end();
  }
});
srv.listen(3002);
```

Postgres exporter — optional custom query (pgvector visibility).

Listing 136: postgres_exporter/queries.yaml

```
pgvector_knn:
```

```
  query: |
    SELECT
      sum(total_time) AS total_ms,
      sum(calls) AS calls
    FROM pg_stat_statements
    WHERE query ILIKE '%<=>%' OR query ILIKE '%vector%';
  metrics:
    - total_ms:
        usage: "GAUGE"
        description: "Total time spent in vector/knn-like queries (ms)"
    - calls:
        usage: "GAUGE"
        description: "Number of vector/knn-like calls"
```

Add environment variable: `PG_EXPORTER_EXTEND_QUERY_PATH=/etc/queries/queries.yaml`.

Grafana — add data source & import dashboards.

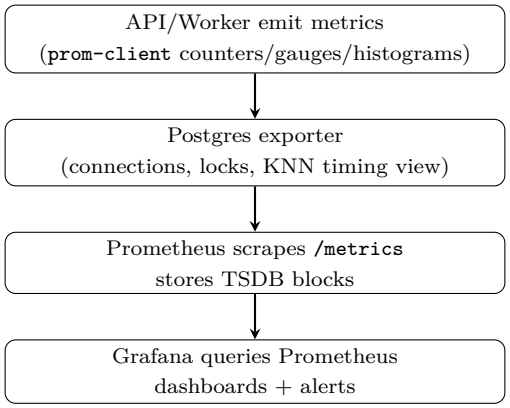
Listing 137: Grafana quick start

```
# open http://localhost:3000 (admin/admin)
# Add Data Source -> Prometheus -> URL: http://prometheus:9090 -> Save & Test
# Import dashboards:
# - 3662 (Node Exporter Full)
# - 19002 (Postgres Exporter)
# - Create a custom API/Worker dashboard using:
#   job:http_p95_ms:5m, http_requests_total, worker_queue_waiting
```

Security & production.

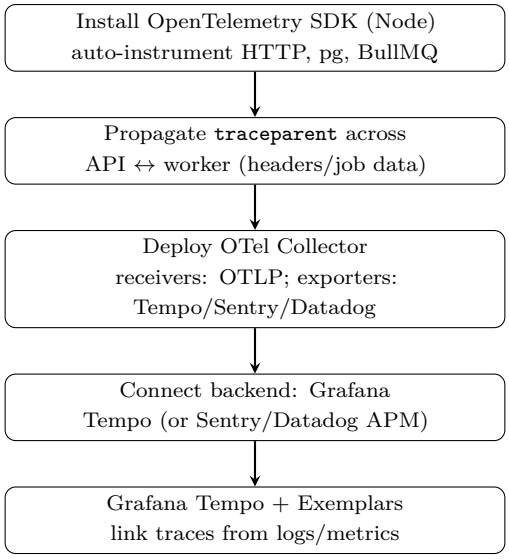
- Place Prometheus behind a private network/VPN or a reverse proxy with auth (and HTTPS).
- Use *read-only* DB credentials for the Postgres exporter.
- Keep label cardinality low (route templates, not raw paths/IDs).
- For long-term retention, enable `remote_write` (Grafana Cloud, Thanos, or VictoriaMetrics).

Metrics — runtime data flow.



15.1.7.4 Tracing (OpenTelemetry)

Setup Workflow



docker-compose.yml (Tempo + Collector + Grafana)

Listing 138: docker-compose.yml

```
version: "3.8"

services:
  tempo:
    image: grafana/tempo:2.5.0
    command: [ "-config.file=/etc/tempo.yaml" ]
    volumes:
      - ./tempo/tempo.yaml:/etc/tempo.yaml:ro
      - tempo-data:/var/tempo
    ports: [ "3200:3200" ]  # Tempo HTTP API for Grafana
    restart: unless-stopped

  otel-collector:
    image: otel/opentelemetry-collector:0.101.0
    command: [ "--config=/etc/otelcol.yaml" ]
    volumes:
      - ./otel/otelcol.yaml:/etc/otelcol.yaml:ro
    ports:
      - "4317:4317"  # OTLP gRPC
      - "4318:4318"  # OTLP HTTP
      - "8888:8888"  # Collector metrics/debug
    depends_on: [ tempo ]
    restart: unless-stopped

```

```
  grafana:
    image: grafana/grafana:10.4.5
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin   # change in prod
    ports: [ "3000:3000" ]
    volumes:
      - grafana-data:/var/lib/grafana
    depends_on: [ tempo ]
    restart: unless-stopped

volumes:
  tempo-data:
  grafana-data:
```

Tempo config — `./tempo/tempo.yaml`

Listing 139: ./tempo/tempo.yam

```
server:
  http_listen_port: 3200

distributor:
  receivers:
    otlp:
      protocols:
        http:
        grpc:

compactor:
  compaction:
    block_retention: 168h  # 7 days

storage:
  trace:
    backend: local
    local:
      path: /var/tempo
```

Collector config — `./otel/otelcol.yaml`

Listing 140: ./otel/otelcol.yaml

```
receivers:
  otlp:
    protocols:
```

```
      http:
      grpc:

processors:
  memory_limiter:
    check_interval: 1s
    limit_percentage: 75
    spike_limit_percentage: 15
  batch:
    send_batch_size: 1024
    timeout: 5s
  resource:
    attributes:
      - key: service.environment
        action: upsert
        value: dev  # set per environment

exporters:
  otlphttp/tempo:
    endpoint: http://tempo:4318
    tls:
      insecure: true

extensions:
  health_check: {}

service:
  extensions: [health_check]
  pipelines:
    traces:
      receivers: [otlp]
      processors: [memory_limiter, batch, resource]
      exporters: [otlphttp/tempo]
```

App instrumentation (Node) — API (NestJS/Express)

```
pnpm add @opentelemetry/sdk-node \
         @opentelemetry/auto-instrumentations-node \
         @opentelemetry/exporter-trace-otlp-http \
         @opentelemetry/propagator-w3c \
         @opentelemetry/instrumentation-express \
         @opentelemetry/instrumentation-http \
         @opentelemetry/instrumentation-pg
```

Listing 141: apps/api/src/otel.ts

```
import { NodeSDK } from '@opentelemetry/sdk-node';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
import { W3CTraceContextPropagator } from '@opentelemetry/core';
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
import { Resource } from '@opentelemetry/resources';

const exporter = new OTLPTraceExporter({
  url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
});

const sdk = new NodeSDK({
  traceExporter: exporter,
  textMapPropagator: new W3CTraceContextPropagator(),
  instrumentations: getNodeAutoInstrumentations({
    '@opentelemetry/instrumentation-http': { enabled: true },
    '@opentelemetry/instrumentation-express': { enabled: true },
    '@opentelemetry/instrumentation-pg': { enabled: true },
  }),
  resource: new Resource({
    'service.name': 'api',
    'service.namespace': 'bidion',
  }),
});

sdk.start();
console.log('[OTel] API tracing enabled');
process.on('SIGTERM', () => sdk.shutdown());
```

Listing 142: API env

```
export OTEL_EXPORTER_OTLP_ENDPOINT=http://otel-collector:4318
```

App instrumentation (Node) — Worker (BullMQ manual spans)

Listing 143: apps/api/src/queues.ts (inject traceparent)

```
import { context, propagation } from '@opentelemetry/api';
import { Queue } from 'bullmq';

export async function enqueueScore(rfpId: string, orgId: string, q: Queue) {
  const carrier: Record<string, string> = {};
  propagation.inject(context.active(), carrier); // writes traceparent
  await q.add('score', { rfpId, orgId, otel: carrier }, { attempts: 5 });
}
```

Listing 144: apps/worker/src/otel.ts

```ts
import { NodeSDK } from '@opentelemetry/sdk-node';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';
import { W3CTraceContextPropagator } from '@opentelemetry/core';
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
import { Resource } from '@opentelemetry/resources';

const exporter = new OTLPTraceExporter({
  url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
});

export const workerSdk = new NodeSDK({
  traceExporter: exporter,
  textMapPropagator: new W3CTraceContextPropagator(),
  instrumentations: getNodeAutoInstrumentations({}),
  resource: new Resource({
    'service.name': 'worker',
    'service.namespace': 'bidion',
  }),
});
workerSdk.start();
```

Listing 145: apps/worker/src/score.processor.ts (extract + span)

```ts
import { context, propagation, trace } from '@opentelemetry/api';

export async function handleScore(job: any) {
  const carrier = (job.data && job.data.otel) || {};
  const parentCtx = propagation.extract(context.active(), carrier);

  return await context.with(parentCtx, async () => {
    const tracer = trace.getTracer('worker');
    return await tracer.startActiveSpan('score-matches', async (span) => {
      try {
        span.setAttribute('rfp.id', job.data.rfpId);
        span.setAttribute('org.id', job.data.orgId);
        // ... KNN, keyword/rules, write matches
      } catch (e) {
        span.recordException(e as Error);
        span.setStatus({ code: 2, message: 'error' });
        throw e;
      } finally {
        span.end();
      }
    });
```

```
  });
}
```

Grafana — add Tempo data source

1. Open `http://localhost:3000` (admin/admin).
2. *Data Sources* → *Add data source* → **Tempo**.
3. URL: `http://tempo:3200` → *Save & Test*.
4. Explore: filter by `service.name` ∈ {`api`, `worker`}.

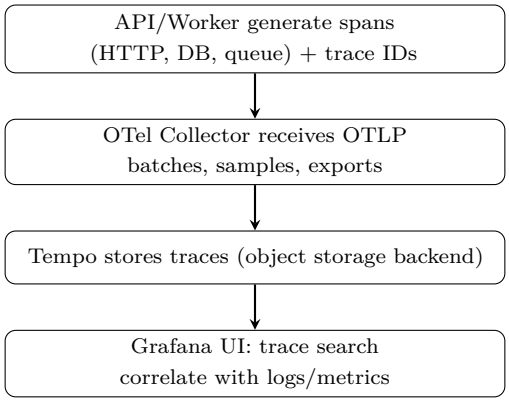
Log/Metric correlation (optional)

- Emit `trace_id`/`span_id` in logs (pino hook) to enable "View trace" from Grafana logs (Loki).
- Export latency histograms with exemplars (Prometheus) to deep-link slow bins to traces.

Security & tenancy

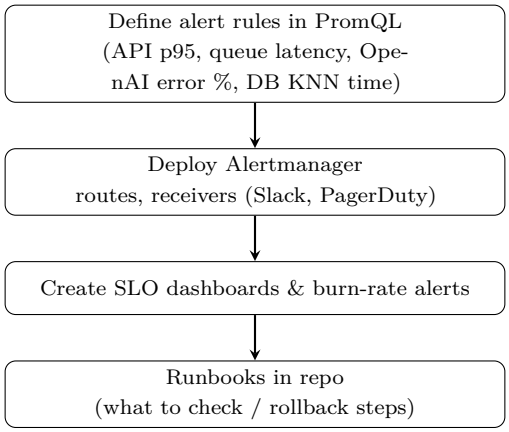
- Put Grafana/Tempo behind TLS and auth (reverse proxy or Grafana OAuth).
- Limit Collector ports (4317/4318) to private networks; do not expose publicly.
- Add `org.id` as a span/resource attribute for per-tenant filtering.

Tracing — runtime data flow.



15.1.7.5 Alerting / On-call

Setup Workflow.



Prometheus is already setup and scraping API, Worker, and Postgres exporters. This section adds: (1) recording/alert rules, (2) Alertmanager routing (Slack/PagerDuty), (3) SLO burn-rate alerts, and (4) on-call runbooks. We *do not* repeat Prometheus setup here.

Recording rules (helper series)

Files: `./prometheus/rules/recording.rules.yml`

Listing 146: prometheus/rules/recording.rules.yml

```
groups:
- name: recording.rules
  interval: 30s
  rules:
  # API: request rate and error rate
  - record: job:http_requests_total:rate1m
    expr: sum by (service, route, method, status) (rate(http_requests_total[1m]))

  - record: job:http_requests_errors_total:rate5m
    expr: sum by (service) (rate(http_requests_total{status=~"5.."}[5m]))

  # API: p95 latency over 5m (requires http_request_duration_seconds_bucket)
  - record: job:http_request_p95_seconds:5m
    expr: |
      histogram_quantile(
        0.95,
        sum by (service, le) (rate(http_request_duration_seconds_bucket[5m]))
      )

  # Queue backlog / latency (export gauges from Worker)
  - record: job:queue_latency_seconds:5m
    expr: max by (queue, service) (bullmq_queue_latency_seconds)

  - record: job:queue_backlog:5m
    expr: max by (queue, service) (bullmq_queue_backlog)

  # OpenAI error ratio (export totals+errors)
  - record: job:openai_error_ratio:5m
    expr: |
      (sum by (service) (rate(openai_requests_errors_total[5m])) /
       clamp_min(sum by (service) (rate(openai_requests_total[5m])), 1))

  # DB pgvector KNN p95 (requires db_knn_seconds_bucket)
  - record: job:db_knn_p95_seconds:5m
    expr: |
      histogram_quantile(
        0.95,
```

```
        sum by (le) (rate(db_knn_seconds_bucket[5m]))
      )
```

Alert rules (concrete thresholds)

Files: `./prometheus/rules/alerts.rules.yml`

Listing 147: prometheus/rules/alerts.rules.yml

```
groups:
- name: service.alerts
  interval: 30s
  rules:
  - alert: APIHighP95Latency
    expr: job:http_request_p95_seconds:5m{service="api"} > 0.5
    for: 10m
    labels: {severity: page, team: platform, component: api}
    annotations:
      summary: "API p95 latency is high"
      description: "p95 over last 5m is {{ $value | printf \"%.3f\" }}s (>0.5s). Check slow routes/DB."

  - alert: QueueBacklogHigh
    expr: job:queue_backlog:5m{queue="rfp-jobs"} > 500
    for: 15m
    labels: {severity: page, team: data, component: worker}
    annotations:
      summary: "Queue backlog high (rfp-jobs)"
      description: "Backlog > 500 for 15m. Scale workers or fix stuck jobs."

  - alert: QueueLatencyHigh
    expr: job:queue_latency_seconds:5m{queue="rfp-jobs"} > 60
    for: 10m
    labels: {severity: page, team: data, component: worker}
    annotations:
      summary: "Queue end-to-end latency high"
      description: "Latency > 60s for 10m. Check Redis, concurrency, batch sizes."

  - alert: OpenAIErrorRateHigh
    expr: job:openai_error_ratio:5m{service="api"} > 0.1
    for: 10m
    labels: {severity: ticket, team: platform, component: api}
    annotations:
      summary: "OpenAI error ratio > 10%"
      description: "5m failure ratio {{ $value | printf \"%.2f\" }}. Check keys, limits, retries/backoff."
```

```
36
37    - alert: DBKNNLatencyHigh
38      expr: job:db_knn_p95_seconds:5m > 0.2
39      for: 15m
40      labels: {severity: ticket, team: data, component: postgres}
41      annotations:
42        summary: "DB KNN p95 > 200ms"
43        description: "pgvector KNN p95 {{ $value | printf \"%.3f\" }}s. Tune indexes/HNSW/
      CPU/IO."
```

SLO Burn-rate alerts (multi-window, multi-burn)

Assume availability SLO: $\geq 99.9\%$ over 30 days (error budget $0.1\% = 0.001$).

Files: `./prometheus/rules/slo-burn.rules.yml`

Listing 148: prometheus/rules/slo-burn.rules.yml

```
1  groups:
2  - name: slo.burn.alerts
3    interval: 30s
4    rules:
5    - record: api:error_ratio:5m
6      expr: |
7        (sum(rate(http_requests_total{service="api",status=~"5.."}[5m])) /
8         clamp_min(sum(rate(http_requests_total{service="api"}[5m])), 1))
9    - record: api:error_ratio:30m
10     expr: |
11       (sum(rate(http_requests_total{service="api",status=~"5.."}[30m])) /
12        clamp_min(sum(rate(http_requests_total{service="api"}[30m])), 1))
13   - record: api:error_ratio:1h
14     expr: |
15       (sum(rate(http_requests_total{service="api",status=~"5.."}[1h])) /
16        clamp_min(sum(rate(http_requests_total{service="api"}[1h])), 1))
17   - record: api:error_ratio:6h
18     expr: |
19       (sum(rate(http_requests_total{service="api",status=~"5.."}[6h])) /
20        clamp_min(sum(rate(http_requests_total{service="api"}[6h])), 1))
21
22   # 99.9% SLO ⇒ budget = 0.001
23   - alert: APISLOFastBurn
24     expr: (api:error_ratio:5m > 14.4 * 0.001) and (api:error_ratio:1h > 14.4 * 0.001)
25     for: 5m
26     labels: {severity: page, slo: "api-availability-99.9-30d"}
27     annotations:
28       summary: "API SLO FAST burn (5m & 1h) > 14.4×"
29       description: "Rapid budget burn. 5m={{ $value | printf \"%.3f\" }}; confirm with 1h
```

```
    ."
30
31  - alert: APISLOSlowBurn
32    expr: (api:error_ratio:30m > 6 * 0.001) and (api:error_ratio:6h > 6 * 0.001)
33    for: 15m
34    labels: {severity: ticket, slo: "api-availability-99.9-30d"}
35    annotations:
36      summary: "API SLO SLOW burn (30m & 6h) > 6×"
37      description: "Sustained degradation. Investigate; consider rollback/feature flags."
```

Alertmanager config (Slack, PagerDuty, routing)

Files: `./alertmanager/alertmanager.yml`

Listing 149: alertmanager/alertmanager.yml

```
1  global:
2    resolve_timeout: 5m
3
4  route:
5    receiver: "slack-default"
6    group_by: ["alertname","cluster","service","component"]
7    group_wait: 30s
8    group_interval: 5m
9    repeat_interval: 4h
10   routes:
11     - matchers: [severity="page"]
12       receiver: "pagerduty"
13       group_wait: 10s
14       repeat_interval: 1h
15     - matchers: [team="data"]
16       receiver: "slack-data"
17
18 receivers:
19   - name: "slack-default"
20     slack_configs:
21       - api_url: "${SLACK_WEBHOOK_URL}"
22         send_resolved: true
23         title: "{{ .CommonLabels.alertname }} ({{ .Status }})"
24         text: |
25           *Summary:* {{ (index .Alerts 0).Annotations.summary }}
26           *Details:* {{ (index .Alerts 0).Annotations.description }}
27           *Labels:* {{ .CommonLabels }}
28           *Silence:* <{{ .ExternalURL }}/#/silences>
29           *Alerts:* {{ range .Alerts }} - {{ .Labels }} {{ end }}
30
```

```
  - name: "slack-data"
    slack_configs:
      - api_url: "${SLACK_WEBHOOK_URL}"
        channel: "#data-oncall"
        send_resolved: true
        title: "{{ .CommonLabels.alertname }} ({{ .Status }})"
        text: |
          *Data Alert:* {{ (index .Alerts 0).Annotations.summary }}
          *Details:* {{ (index .Alerts 0).Annotations.description }}

  - name: "pagerduty"
    pagerduty_configs:
      - routing_key: "${PAGERDUTY_ROUTING_KEY}"
        severity: "{{ if eq .CommonLabels.severity \"page\" }}critical{{ else }}error{{
    end }}"
        send_resolved: true

inhibit_rules:
  - source_matchers: [severity="page"]
    target_matchers: [severity="ticket"]
    equal: ["alertname","service"]
```

App instrumentation (what to expose)

- **API**: `http_request_duration_seconds_bucket`, `http_requests_total{status}`
- **Worker/Queue**: `bullmq_queue_latency_seconds`, `bullmq_queue_backlog`
- **OpenAI**: `openai_requests_total`, `openai_requests_errors_total`
- **DB/KNN**: `db_knn_seconds_bucket`

Runbook examples (checked into repo)

Files: `./runbooks/APIHighP95Latency.md`

Listing 150: runbooks/APIHighP95Latency.md

```
# APIHighP95Latency
## Triage
1) Grafana API dashboard → "Latency p95 by route".
2) Check 5xx spikes, deploys (last 1h), DB CPU/IO.
## Mitigation
- Scale API/Worker +1.
- Rollback last deploy (<30m).
- Toggle feature flags to reduce load.
## Deep-dive
- Offending route labels.
```

```
- Postgres slow queries, pgvector indexes (HNSW params).
- OpenAI limits/errors.
## Closure
- Postmortem issue with timeline + actions.
```

Files: `./runbooks/QueueBacklogHigh.md`

Listing 151: runbooks/QueueBacklogHigh.md

```
# QueueBacklogHigh
## Triage
1) Worker dashboard → backlog & latency.
2) Redis health, BullMQ concurrency, stalled jobs.
## Mitigation
- +2 temp concurrency.
- Drain DLQ/poison.
- Pause producers for stuck job types.
## Deep-dive
- Recent changes to parse-doc/embed-chunks/score-matches.
- Redis CPU/mem, worker node network.
## Closure
- Tune batch size/retries; add per-job timeouts.
```

Validation & Ops commands

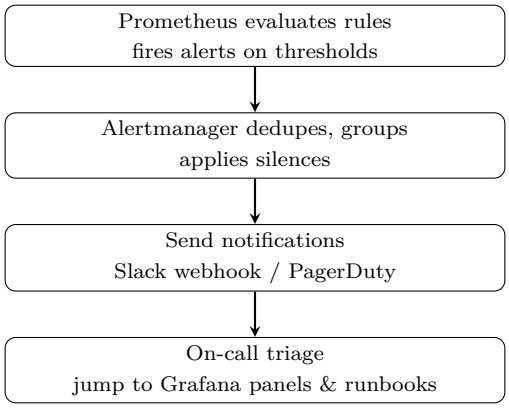
Listing 152: Validate rules and AM config

```
# Validate Prom rules:
docker run --rm -v $PWD/prometheus:/etc/prom prom/prometheus \
  promtool check rules /etc/prom/rules/*.yml

# Validate Alertmanager config:
docker run --rm -v $PWD/alertmanager:/etc/am prom/alertmanager:latest \
  amtool check-config /etc/am/alertmanager.yml

# Reload Prometheus (requires --web.enable-lifecycle already set in Prom setup):
curl -X POST http://localhost:9090/-/reload
```

Notes & Threshold Tuning

- Page only for user-visible impact; file tickets for noise.
- Calibrate API p95 by historical p95/p99; 500 ms is a placeholder.
- For KNN latency, watch p95/p99 under load; consider HNSW and `work_mem`.
- Align SLOs with business impact (e.g., 99.9% vs. 99.5%).

Alerting / On-call — runtime data flow.



Operational Remarks.

- **PII hygiene:** redact at the app (`pino` redact), not only in sinks. Avoid logging request bodies unless necessary.
- **Correlation:** always stamp `x-request-id` and propagate `traceparent`; include IDs in logs and (sparingly) as Prometheus labels.
- **DB visibility:** create a tiny view that times KNN queries and export as a Prometheus gauge; directly tracks vector search health.
- **Cost control:** start managed (Grafana Cloud, Sentry). If we outgrow, move to self-hosted Loki/Prometheus/Tempo with object storage.
- **Runbooks:** keep Markdown runbooks in-repo and link them in alert annotations (`runbook_url`) to reduce MTTR.

15.1.8 Upgrade Paths (Post-MVP)

- Move API/Workers from Fly/Render to AWS ECS/Fargate (or Kubernetes) when we need VPC/IAM controls or higher scale.
- Switch `ivfflat` to `HNSW` in pgvector for better recall/latency; add read replicas if KNN load increases.
- Introduce a feature-flag service and per-tenant rate limits; add SSO (OIDC) and scoped API tokens.

15.2 Scaling Plan (MVP → v1)

Goals.

Sustain *millions of users / org seats* with predictable SLOs (API p95 $\leq$ 500 ms; KNN p95 $\leq$ 200 ms under nominal load), while keeping cost linear with usage. *Planning envelope (for 1M users):* peak **1,000–1,500 RPS** on API, **10–30M** total embedding vectors (3,072 dims), **3–5k jobs/min** worker throughput, primary DB **0.5–1.5 TB**.

15.2.1 Horizontally scale stateless tiers first.

API (NestJS) — Replicas, Concurrency, Pooling

Scale behind an LB. Start *6–10 replicas* (1 vCPU, 1–2 GB) and HPA on p95/RPS. Connection pool via `pgbouncer` (*pool_size 150–250*, `max_client_conn 2,000`), keep per-pod DB connections low (*20–30* `MAX_CONNECTIONS` / instance).

Variable	1M Target	Notes
Replicas	6–10 pods	Behind LB; HPA on p95/RPS
Pod size	1 vCPU / 1–2 GB	Increase only if CPU-bound
Per-pod DB conns	20–30	Keep total active < 500 via pooling
pgBouncer pool_size	150–250	Transaction pooling mode
pgBouncer max_client_conn	2,000	Headroom for spikes
HTTP timeouts	3–5 s	With retries + circuit breakers
Health/readiness	`/healthz` every 10s	Max surge 1, max unavailable 0

Workers (BullMQ) — Concurrency, Autoscale, Limits

Separate deployments per queue (`parse-doc`, `embed-chunks`, `score-matches`). Per-pod concurrency: *parse=4*, *embed=16*, *score=8*; autoscale replicas *4→20* on backlog $>$ *1,000 jobs for 5 min*. Cap LLM egress $\leq$ *5 QPS/org* and global *50–100 QPS*.

Variable	1M Target	Notes
Deployments	per-queue (parse/embed/score)	Isolate lanes to tune independently
Per-pod concurrency	parse=4; embed=16; score=8	Cap to protect DB/LLM
Replicas	$4 \rightarrow 20$	Scale when backlog $>$ 1,000 for 5 min
Retry/backoff	exp. 250 ms $\rightarrow$ 5 s + jitter	Max 5 attempts; DLQ on fail
Per-org LLM QPS	≤ 5 QPS	Noisy-neighbor guardrail
Global LLM QPS	50–100 QPS	Align with provider quota/budget
Max inflight/org	500 jobs	Backpressure before saturation

Edge / CDN — Caching and Freshness

cache public GETs w/ `Cache-Control: public, max-age=`*120*`, s-maxage=`*300*. For app APIs, use `stale-while-revalidate=`*60* on safe endpoints (e.g., cached `/rfps/:id/matches`).

Header/Setting	Value	Notes / Scope
Cache-Control (public GET)	`public, max-age=120, s-maxage=300`	Static/public assets
SWR (safe API reads)	`stale-while-revalidate=60`	e.g., `/rfps/:id/matches` (pre-computed)
ETag/If-None-Match	enabled	Validate freshness cheaply
Compression	gzip/br	Ensure on CDN + origin
CDN regions	multi-region POPs	Default provider footprint OK

15.2.2 Postgres scaling path (Supabase DB).

Knob	1M Target	Notes
Instance size	8 vCPU / 32 GB (to 16/64)	Vertical headroom first; watch CPU/IO wait $< 5\%$
Provisioned IOPS	$\geq$ 10k	Ensure disk class supports target IOPS
Pooling mode	pgBouncer (transaction)	Set drivers `prepared_statements=false` if needed
Active backends	< 500	Keep via pooling + low per-pod conns
Partitions (org)	HASH 32–64	On `org_id` for large shared tables
Partitions (time)	MONTHLY	For `embeddings`/`chunks`
Read replicas	1–2	Route dashboards/analytics; lag < 2 s
Backfills	5k–20k rows/txn	Off-peak; throttle to keep p95 budget
Online reindex	`REINDEX CONCURRENTLY`	Avoid table locks
RLS strategy	shared table + `org_id`	Consider per-tenant partitions for hot orgs

- **Vertical headroom first**: target *8 vCPU / 32 GB RAM* (scale to *16 vCPU / 64 GB* as needed), provisioned IOPS $\geq$ *10k*. Watch I/O wait ($<$ *5%*), WAL rate, autovacuum.
- **Connection pooling**: `pgbouncer` in *transaction* mode; set drivers `prepared_statements=false` if required. Keep total active backend conns $<$ *500*.
- **Partitioning**: partition large tables by `org_id` (HASH *32–64 partitions*) and/or time (MONTHLY for `embeddings`/`chunks`). Ensure constraint exclusion/pruning on queries.
- **Read replicas**: *1–2* replicas; route dashboards/analytics and read-heavy endpoints to replicas. Accept replica lag $<$ *2 s*.
- **Online schema changes**: `ADD COLUMN` nullable, backfill batches (*5k–20k rows/txn*), `REINDEX CONCURRENTLY`. Avoid peak-time table locks.
- **RLS at scale**: shared tables with `org_id` + RLS (see §14); consider per-tenant partitions for hot orgs ($< 5\%$ causing $> 50\%$ load).

15.2.3 Vector search (pgvector) scale knobs.

Parameter	1M Target	Notes
Index type	Start IVFFLAT; move to HNSW	Switch after write rate stabilizes
IVFFLAT lists	2,000–4,000	For 10–30M vectors
IVFFLAT probes	8–16	Balance recall vs latency
HNSW M	16	Out-degree; memory/recall tradeoff
HNSW ef_construction	200	Build-time accuracy
HNSW ef_search	64–128	Increase for higher recall
Query k	50	Candidates from ANN
Shortlist m	200	Re-rank domain scoring; keep KNN p95 $\leq$ 200 ms
Maintenance	ANALYZE nightly; VACUUM as needed	HNSW bulk-rebuilds off-peak

- **Index choice**: start **IVFFLAT**; switch to **HNSW** once write rate stabilizes.
- **IVFFLAT**: `lists` $\approx$ *2,000–4,000* for *10–30M* vectors; `probes` *8–16*.

- **HNSW**: `M=16`, `ef_construction=200`, `ef_search=64-128` (increase for recall).
- **Recall/latency**: query $k = 50$; shortlist top $m = 200$ for re-rank (domain scoring) to keep KNN p95 ≤ 200 ms.
- **Maintenance**: `ANALYZE` nightly; `VACUUM` as needed; schedule HNSW bulk rebuilds off-peak.

15.2.4 Queues, backpressure, and cost controls.

Control	1M Target	Notes
Per-org rate limits	embed 60/min/org; score 120/min/org	Tune by SLO/cost
Global LLM QPS	50–100 QPS	Respect provider quota
Max inflight/org	500 jobs	Backpressure before saturation
Batch size (embeds)	64–128 texts/call	Model-dependent max
Backoff	250 ms → 5 s + jitter	Max 5 attempts; circuit open 60 s
Idempotency keys	`chunk_hash`, `document_id`	`matches`: delete+insert pattern
Priority lanes	Interactive > bulk	Bulk backfills rate limit 10/min/org

- **Backpressure**: per-queue rate limits (*embed 60/min/org*, *score 120/min/org*); max inflight per org *500*. Global caps aligned with LLM QPS budget.
- **Idempotency**: dedupe by `chunk_hash` and `document_id`; use *delete+insert* for `matches`.
- **LLM cost guards**: batch embeddings *64–128 texts/call*, exponential backoff (base *250 ms*, max *5 s*, jitter), circuit breaker open *60 s*.
- **Priority lanes**: interactive queues high-priority; bulk backfills low-priority with *rate limit 10/min/org*.

15.2.5 Caching layers.

Layer	1M Target	Notes
HTTP cache TTL	60–300 s	CDN + `ETag`/`Cache-Control`
Redis cluster	3 nodes, 3–6 GB each	Managed; private networking
Hot keys	< 5M total	Eviction policy: allkeys-lru
RFP matches TTL	30–120 s	Memoize top matches per RFP
App LRU cache	500–2,000 entries	Small reference data (weights/rules)

- **HTTP cache**: CDN + `ETag`/`Cache-Control`; typical TTL *60–300 s*.
- **Redis**: 1 small cluster (*3 nodes, 3–6 GB each*); memoize top matches per RFP (*TTL 30–120 s*); keep hot key count < *5M* total.
- **App cache**: in-process LRU *500–2,000 entries* for small reference data (weights/rules).

15.2.6 Multi-region (v1+).

Aspect	1M Target	Notes
Stateless topology	Active/active per region	API/workers only
Database	1 primary + 1–2 read replicas	Eventual consistency on non-critical reads
Org placement	Pin to home region	Reduce KNN latency (data gravity)
Routing	GeoDNS / latency-based	Sticky to home region
DR	PITR + cross-region backups	Failover rehearsal quarterly

- **Topology**: active/active stateless per region; Postgres primary in one region + *1–2* read replicas elsewhere (eventual consistency for non-critical reads).
- **Data gravity**: pin orgs to a home region; route with GeoDNS.
- **DR**: PITR, cross-region backups; quarterly failover rehearsal.

15.2.7 Migrations and deploy safety.

Control	1M Target	Notes
Artifact promotion	Same SHA (staging→prod)	Blue/green or rolling
Health gate	$> 99\%$ success over 5 min	Gate rollout by p95 + error rate
Safe schema change	Add NULLable; dual-write; dual-read	Backfill 5k–20k rows/txn off-peak
Feature flags	$5\% \to 25\% \to 100\%$	Instant rollback path

- **Immutable artifacts**: promote same SHA (staging → prod); rolling/blue-green; pod P95 health checks (*success > 99% over 5 min*) gate rollout.
- **Safe migrations**: write to new nullable column, dual-read phase, async backfill (*5k–20k rows/txn*), flip reads, drop legacy.
- **Feature flags**: progressive rollout *5%→25%→100%*; instant rollback.

15.2.8 Observability & SLO enforcement.

Signal / Policy	1M Target	Action
API p95	≤ 500 ms	HPA scale-out; throttle producers
KNN p95	≤ 200 ms	Tune k, m, probes/ef_search; add replicas
Worker backlog	< 10k jobs	Scale workers first; then API
Burn-rate alerts	5m/1h & 30m/6h	Page + autoscale; apply rate limits
Cache hit rate	$> 80\%$ (hot endpoints)	Increase TTLs; widen caching scope
Cost dashboards	Per-org LLM spend	Enforce QPS budgets

- **Golden signals**: API latency/error rate, worker latency/throughput, DB CPU/IO/locks, KNN p95/p99.
- **Burn-rate alerts**: 5m/1h & 30m/6h (see Alerting). On breach: auto-scale workers, then API; throttle producers if backlog > *10k* or p95 budget violated.
- **Cost dashboards**: LLM spend/org, embeddings/day, cache hit rates (> *80% for hot endpoints*).

15.2.9 Data lifecycle.

Data Class	1M Target	Notes
Deleted docs	Purge within 24 h	Remove embeddings/chunks + caches
Cold data	Archive after 90 d	Move to object storage (S3/Supabase)
GDPR deletes	Audit $<$ 30 d	Hard-delete; background scrub of vectors
TTL policies	Keys expire by default	Prevent unbounded growth

- **TTL/archival**: purge embeddings/chunks for deleted docs within *24 h*; archive cold data to object storage after *90 d*.
- **GDPR/tenant deletes**: hard-delete with background scrub of vectors and caches; audit in $<$ *30 d*.

Summary.

1. **Shed load before fall over**: global and per-org rate limits; circuit breakers for OpenAI/DB (open *60 s*, half-open after *10* successes).
2. **Thin request path**: precompute `matches`; `GET /rfps/:id/matches` must be read-only (*p95* $\leq$ *200 ms from cache/replica*).
3. **Elasticity**: scale workers on backlog; API on RPS/p95; cap per-pod concurrency with budgets (embed *16*, score *8*).
4. **Partition first, shard later**: partitions (*32–64*) first; shard only if partitions + replicas are insufficient (hot orgs, TB-scale).
5. **Test**: k6 thresholds in CI (p95 $<$ *500 ms*, fail rate $<$ *2%*); nightly soak; chaos drills for Redis/DB/LLM brownouts.

15.3 Maintenance, Resilience & Recovery

15.3.1 Postgres: Backups, PITR, and Maintenance

Intuitively.

- **Backups are like photo albums.** We take regular "pictures" of the database so we can look back if something goes wrong.
- **PITR is a time machine.** Point-In-Time Recovery lets us rewind the database to *exactly* how it looked a few minutes ago—like scrubbing a video back to the moment before a mistake.
- **RPO = how much we can lose.** If our RPO is $\leq$5 minutes, the worst-case is we lose at most five minutes of new changes.
- **RTO = how long we're down.** If our RTO is $\leq$60 minutes, that's our goal for how fast we can restore and get the app talking to the database again.
- **Maintenance = chores.** `ANALYZE` is updating the map so Postgres knows where things are; `VACUUM` is taking out the trash. We schedule these chores so the house stays tidy and fast.

Rationale.

- **Why PITR?** Accidents happen—bad migrations, deletes, or bugs. PITR gives us a safe rewind button instead of starting from scratch.

- **Why RPO/RTO targets?** Clear targets tell us the *max data we might lose* and the *max time to recover*. They keep the team honest and drills measurable.
- **Why scheduled maintenance?** Small, regular cleanups (`VACUUM/ANALYZE`) beat rare, heavy cleanups. The app stays fast, and we avoid surprise slowdowns from bloat or stale stats.
- **Why verify with drills?** Practicing restores proves our backups are usable, our cutover steps work, and that we really meet the RPO/RTO we promised.

Control	MVP Target	Notes / Basis
Backup model	Managed PITR enabled	Continuous WAL archiving + base backups (managed). [1]
RPO (data loss window)	$\leq$ 5 minutes	Supabase advertises near-real-time WAL shipping; verify during drills. [1]
RTO (restore time)	$\leq$ 60 minutes	Includes restore to timestamp + app re-point + checks. [2]
Retention	7–14 days PITR	Balance cost vs. rollback needs. [1]
Maintenance windows	Weekly VACUUM/ANALYZE	Keep bloat low; schedule with `pg_cron`. [4, 3]

15.3.1.1 Setup & Test PITR (Managed Supabase).

1. **Enable PITR in project settings.** In Supabase, turn on Point-in-Time Recovery (keeps continuous WAL + base backups). *Record the retention window.* [1]
2. **Create a drill marker.** Run a tiny, unique transaction so we can target a known timestamp:

```
-- Mark a drill time (note the server timestamp)
CREATE TABLE pitr_drill_mark(id int primary key, note text);
INSERT INTO pitr_drill_mark VALUES (1, 'drill: 2025-03-01T10:30Z');
SELECT NOW(); -- capture this as T_marker
```

3. **Simulate an error.** Drop the table or delete rows to mimic data loss:

```
DROP TABLE pitr_drill_mark;
```

4. **Restore to timestamp.** From Supabase PITR UI, *restore to a new database instance* at `T_marker + 1s`. (Postgres PITR replays WAL to a point-in-time.) [2]
5. **Validate.** Connect to the restored DB and confirm the marker exists:

```
SELECT * FROM pitr_drill_mark;  -- should return the inserted row
```

6. **Cutover plan.** Prepare app `DATABASE_URL` switching (feature flag or secret rotate). Practice switching API/workers to the restored DB and back.

15.3.2 Postgres: Scheduling, Vacuum/Analyze, and Tuning

Intuitively.

Think of **`pg_cron`** as a calendar *inside Postgres* that runs SQL on a schedule.[1]

[1] pg_cron: cron-based job scheduler for Postgres & Supabase Cron (backed by `pg_cron`).

- **ANALYZE every night:** The database keeps a "map" of what our data looks like so it can find things fast. `ANALYZE` refreshes planner statistics so query plans stay accurate. Doing it nightly keeps searches speedy without much disruption.[2]
- **VACUUM (`VERBOSE, ANALYZE`) every week:** As rows get updated/deleted, little bits of trash pile up. `VACUUM` reclaims dead tuples (and with `ANALYZE` also refreshes the map). Weekly is a good routine: often enough to prevent bloat, not so often that it gets in the way.[3]

Rationale.

- **pg_cron jobs = automatic chores.** Schedule `ANALYZE` nightly (quick map update) and `VACUUM` weekly (deep clean) so the planner stays smart and tables don't get bloated.[4]
- **Autovacuum per-table tuning = clean busy rooms more often.** Most rooms (tables) are fine with the default schedule—*autovacuum usually suffices*. For the few "messy" rooms with lots of changes, set stricter triggers so they get cleaned sooner—without making the whole house over-clean.[5] Adjust per-table thresholds *only after telemetry shows bloat or stale stats* (watch `n_dead_tup`, `n_mod_since_analyze`).[6]
- **Operational constraint note.** `VACUUM` ordinarily requires the table *MAINTAIN* privilege (owners can vacuum their DB) and *cannot* be executed inside a transaction block.[7]

15.3.2.1 Pre-requisites (Supabase-managed Postgres)

```
-- Enable the scheduler in the target database
-- (Supabase Dashboard → Database → Extensions → pg_cron)
CREATE EXTENSION IF NOT EXISTS pg_cron;

-- Notes:
-- On Supabase, pg_cron jobs run under the built-in cron role.
-- Typically DO NOT need a separate maintenance role.
-- Prefer simple single-statement jobs for reliability (ANALYZE; VACUUM ...;).
```

15.3.2.2 Pre-requisites (Self-hosted Postgres: install & preload pg_cron)

Listing 153: Install & enable pg_cron (example: PostgreSQL 15 on Ubuntu/Debian)

```
# Install the extension package matching our PG major version
sudo apt update && sudo apt install postgresql-15-cron

# Edit postgresql.conf (shared_preload_libraries requires a restart)
# e.g., /etc/postgresql/15/main/postgresql.conf
```

[2] PostgreSQL `ANALYZE` docs.
[3] PostgreSQL `VACUUM` docs.
[4] pg_cron on GitHub & `ANALYZE`/`VACUUM` docs.
[5] Autovacuum guidance (RDS/Aurora, generally applicable).
[6] See PostgreSQL routine vacuuming guidance and when to tune (docs), and the statistics views/columns to monitor (docs).
[7] `VACUUM` privilege & transaction-block notes.

```
echo "shared_preload_libraries = 'pg_cron'" | sudo tee -a /etc/postgresql/15/main/
    postgresql.conf
echo "cron.database_name = 'your_database'" | sudo tee -a /etc/postgresql/15/main/
    postgresql.conf

# Restart Postgres to apply preload changes
sudo systemctl restart postgresql
```

Listing 154: Create the extension in the target database (self-hosted)

```
-- Connect to our target DB:
CREATE EXTENSION IF NOT EXISTS pg_cron;
```

15.3.2.3 Create jobs (nightly ANALYZE, weekly VACUUM)

```
-- Nightly stats refresh (02:30) - run ANALYZE for the current DB
SELECT cron.schedule(
  'stats-analyze-nightly',
  '30 2 * * *',
  $$ ANALYZE; $$
);

-- Weekly vacuum for hot tables (Sunday 03:00) - start with public.matches
SELECT cron.schedule(
  'vacuum-weekly-matches',
  '0 3 * * 0',
  $$ VACUUM (VERBOSE, ANALYZE) public.matches; $$
);
```

15.3.2.4 Force-run now (staging/CI validation)

```
-- List jobs to get jobid
SELECT jobid, jobname, schedule FROM cron.job ORDER BY jobid;

-- Trigger a run immediately
SELECT cron.run_job(<jobid>);

-- Inspect run history
SELECT jobid, status, return_message, start_time, end_time
FROM cron.job_run_details
ORDER BY start_time DESC
LIMIT 10;
```

15.3.2.5 Test: Observability queries (did it help?)

```
-- Table-level VACUUM/ANALYZE timestamps
SELECT relname, last_vacuum, last_autovacuum, last_analyze, last_autoanalyze
FROM pg_stat_user_tables
WHERE schemaname='public' AND relname IN ('matches')
ORDER BY relname;

-- Bloat/health proxy: live vs dead tuples, scans, n_mod_since_analyze
SELECT
  relname,
  n_live_tup, n_dead_tup,
  n_mod_since_analyze,
  vacuum_count, autovacuum_count,
  analyze_count, autoanalyze_count
FROM pg_stat_user_tables
WHERE schemaname='public' AND relname IN ('matches');
```

15.3.2.6 Autovacuum tuning (targeted)

Apply per-table reloptions (only if churn is high).

```
ALTER TABLE public.matches
  SET (autovacuum_vacuum_scale_factor = 0.05,   -- trigger vacuum at ~5% churn
       autovacuum_analyze_scale_factor = 0.02); -- trigger analyze at ~2% churn
```

Verify reloptions are active.

```
SELECT relname, reloptions
FROM pg_class
WHERE relname='matches';
```

Monitor outcomes after load.

```
SELECT relname, n_dead_tup, vacuum_count, autovacuum_count,
       analyze_count, autoanalyze_count, last_autovacuum, last_autoanalyze
FROM pg_stat_user_tables
WHERE relname='matches';
```

15.3.2.7 Alternative if pg_cron is unavailable (self-hosted)

Listing 155: OS cron / systemd timers calling psql

```
# crontab -e
30 2 * * * PGPASSWORD=... psql "$DATABASE_URL" -v ON_ERROR_STOP=1 -c "ANALYZE;"
0  3 * * 0 PGPASSWORD=... psql "$DATABASE_URL" -v ON_ERROR_STOP=1 \
  -c "VACUUM (VERBOSE, ANALYZE) public.matches;"
```

15.3.2.8 Operational remarks (timeouts & safety)

- If a strict `statement_timeout` is configured, prefer targeted VACUUMs (specific tables) and keep maintenance windows off-peak.
- Start with the hottest tables (e.g., `public.matches`); expand scope as headroom allows.
- Prefer per-table autovacuum tuning before cluster-wide changes on managed platforms.

15.3.2.9 Passed-Results Variables (Verification Matrices)

Postgres

Area	Check / Query	Pass Criteria (Expected)
pg_cron jobs exist	`SELECT jobid, jobname, schedule FROM cron.job;`	At least two rows: `stats-analyze-nightly` @ `30 2 * * *` and `vacuum-weekly-matches` @ `0 3 * * 0`.
pg_cron last run status	`SELECT status FROM cron.job_run_details ORDER BY start_time DESC LIMIT 1;`	`status = 'SUCCEEDED'` (or provider-specific success string).
Forced run works	`SELECT cron.run_job(<jobid>);` then inspect `job_run_details`	Run record appears with recent `start_time`, non-error `return_message`.
Analyze/Vacuum timestamps	`SELECT last_analyze, last_autoanalyze, last_vacuum, last_autovacuum FROM pg_stat_user_tables WHERE relname='matches';`	At least one of the timestamps updated after the forced run window.
Dead tuples trend	`SELECT n_dead_tup FROM pg_stat_user_tables WHERE relname='matches';`	Stable or decreasing after weekly VACUUM; alert if growth is monotonic across days.
Autovacuum reloptions	`SELECT reloptions FROM pg_class WHERE relname='matches';`	Includes `autovacuum_vacuum_scale_factor=0.05, autovacuum_analyze_scale_factor=0.02` .
Autovacuum activity	`SELECT autovacuum_count, autoanalyze_count FROM pg_stat_user_tables WHERE relname='matches';`	Counts increase over time under write load (post-tuning).

15.3.3 Redis: Durability & Recovery (BullMQ backing store)

15.3.3.1 Managed Redis (Redis Cloud by Redis)

Goal: durable BullMQ metadata with pragmatic AOF settings, HA, TLS, and verifiable backups.

15.3.3.1.1 Setup.

1. **Create a Redis Cloud database (single region).**
   (a) Redis Cloud Console → *Subscriptions* → **New Subscription**.
   (b) Choose **Cloud: AWS**, **Region: us-east-1**, **Plan: Fixed** with **1 GB RAM** and **10 GB storage**.
   (c) Name the database `bullmq-prod`. [11]
2. **Enable persistence (AOF) with concrete values.**
   (a) Database → **Data Persistence**.
   (b) Turn on **Append Only File (AOF)** and set **Fsync policy = Every second (`everysec`)**. [15]
   (c) (Optional) Enable **RDB snapshots** at **every 15 minutes**. [15]
   (d) Ensure **Automatic AOF rewrite** remains **Enabled** (default threshold is fine). [15]
3. **High availability (HA).**
   (a) Database settings → **High Availability** → set **Replication: On**.
   (b) Target **1 primary + 1 replica**. [11]
4. **Memory safety (no silent loss).**
   (a) Set **Eviction Policy** to `noeviction`. [16]
   (b) Add alerts at **80% memory** and **80% disk**.
5. **Security (TLS + AUTH).**
   (a) Use the provided `rediss://` endpoint (TLS). [9]
   (b) Rotate the default password; optionally create ACL user `bullmq` with minimal commands.
   (c) Restrict source IPs to API/worker egress via Redis Cloud network rules. [11]
6. **Backups & restore drills.**
   (a) Schedule **Automated Backups**: **daily at 03:00 UTC**, retention **7 days**. [10]
   (b) From *Backups*: **Create database from backup** for drills (name: `bullmq-restore-drill`); point a staging worker and verify keys/jobs. [10]
   (c) Repeat the drill **quarterly**. [10]
7. **Wire BullMQ to Redis Cloud (Node.js).**
   (a) Set:

```
export REDIS_URL="rediss://default:pA$$-exAmplE-9nv@redis-12345.c12.us-east-1-1.
    ec2.redns.redis-cloud.com:12345"
```

(b) Example usage:

```
import { Queue } from 'bullmq';
import IORedis from 'ioredis';

const redis = new IORedis(process.env.REDIS_URL!); // TLS implied by rediss://
export const scoreMatchesQ = new Queue('score-matches', { connection: redis });
```

8. **CLI sanity (TLS)** — from a bastion/CI runner:

```
# Expect PONG (TLS via rediss://)
redis-cli -u "rediss://default:pA$$-exAmplE-9nv@redis-12345.c12.us-east-1-1.ec2.
    redns.redis-cloud.com:12345" PING
```

15.3.3.1.2 Test

Test maintenance, durability & recovery.

1. **AOF durability smoke.**
   (a) Write a marker: `SET bullmq:test:marker "mvp-2025-09-21T10:00Z"`.
   (b) Wait > 1 s (fsync window), then run: `INFO persistence` and verify `aof_enabled:1`. [17]
   (c) Restart the client (not the service) and `GET` the marker; it must persist.
2. **AOF rewrite health.**
   (a) Generate some write load (enqueue 1,000 small jobs/keys).
   (b) Check: `INFO persistence` → `aof_last_bgrewrite_status:ok`, `aof_rewrite_in_progress:0`. [17]
3. **Eviction policy guard.**
   (a) Read config: `CONFIG GET maxmemory-policy` ⇒ `noeviction`. [16]
4. **TLS/auth enforcement.**
   (a) `redis-cli -u rediss://...  PING` ⇒ `PONG`.
   (b) Try wrong password ⇒ expect `NOAUTH Authentication required` / failure.
5. **Backup & restore drill.**
   (a) Ensure a fresh backup exists (scheduled 03:00 UTC).
   (b) In the console, *Create database from backup* ⇒ name `bullmq-restore-drill`.
   (c) Connect staging worker and verify presence of sentinel key `bullmq:test:marker` and representative BullMQ keys (e.g., `bull:score-matches:wait`). [10]

15.3.3.1.3 Results

Passed-results variables (verification matrix).

Area	Check / Command	Pass Criteria (Expected)
AOF enabled & policy	`CONFIG GET appendonly`, `CONFIG GET appendfsync` (UI/CLI)	`appendonly = yes`, `appendfsync = everysec`.
AOF health	`INFO persistence`	`aof_enabled:1`, `aof_last_bgrewrite_status:ok`, `aof_rewrite_in_progress:0`.
Eviction policy	`CONFIG GET maxmemory-policy`	`noeviction`.
TLS/auth	`redis-cli -u rediss://... PING`	`PONG` only when authenticated; TLS negotiated.
Backup & restore drill	Console: "Create database from backup"	New DB online; sentinel key and BullMQ keys present.

15.3.3.1.4 Rationale.

- **AOF every second** → at most ≤ 1 s potential data loss with strong throughput. [15]
- **noeviction** → never silently drop queue keys; fail fast and scale/clean. [16]
- **HA replica** → managed failover with minimal downtime. [11]

15.3.3.1.5 Monitoring.

- **AOF health:** `aof_enabled=1`, `aof_last_bgrewrite_status=ok`, `aof_rewrite_in_progress:0`. [17]
- **Capacity:** memory usage $<$ `80%`, disk usage $<$ `80%`; alert if `used_memory_dataset` growth exceeds plan.
- **Latency:** `latency doctor` in emergencies; otherwise export vendor metrics to your dashboard. [12]

15.3.3.2 Self-hosted on Render (Private Service)

Goal: durable BullMQ metadata with AOF, private networking, explicit backups, and verifiable restore drills.

15.3.3.2.1 Setup.

1. **Create a Private Service with persistent disk.**
   (a) In Render, create a new **Private Service** from your repo (Docker).[13]
   (b) Attach a **Persistent Disk**: **20 GB**, mount path `/data`.[14]

(c) Service name: `redis-selfhosted`; internal hostname will be `redis-selfhosted` on the private network.[19]

2. **Repository files:** add `render.yaml`, `Dockerfile`, `redis.conf`, `scripts/backup-aof.sh`. (Image pin example uses the official Redis Docker image.[21])

Listing 156: render.yaml

```
services:
  - name: redis-selfhosted
    type: private_service
    env: docker
    plan: standard
    autoDeploy: true
    disk:
      name: redis-data
      mountPath: /data
      sizeGB: 20
    envVars:
      - key: REDIS_PASSWORD
        value: pA$$-exAmplE-render
        generateValue: true
    # Render will build Dockerfile at repo root
    # Redis command and requirepass are set by the container entrypoint

```

Listing 157: Dockerfile

```
FROM redis:7.2-alpine
COPY redis.conf /usr/local/etc/redis/redis.conf
COPY scripts/backup-aof.sh /usr/local/bin/backup-aof.sh
RUN chmod +x /usr/local/bin/backup-aof.sh
# Ensure data dir exists (Render mounts /data as a persistent disk)
RUN mkdir -p /data
EXPOSE 6379
# Pass requirepass from env; persist to /data; fsync everysec
ENTRYPOINT ["sh","-c","exec redis-server /usr/local/etc/redis/redis.conf --
    requirepass $REDIS_PASSWORD"]

```

Persistence settings (`appendonly yes`, `appendfsync everysec`) per Redis persistence guidance.[15, 21]

Listing 158: redis.conf (Render)

```
# -------- Persistence --------
dir /data
appendonly yes
appendfsync everysec
```

```
# Optional lightweight snapshot as a secondary safety
save 900 1

# -------- Memory / Eviction --------
maxmemory 0
maxmemory-policy noeviction

# -------- Networking / Hardening --------
protected-mode yes
bind 0.0.0.0
port 6379
# TLS is not configured here; rely on Render private network isolation
# If your org mandates TLS on internal hops, front this Private Service
# with a TLS-terminating proxy (e.g., stunnel/envoy/HAProxy) and point clients at it
    .

# -------- AOF Rewrite (leave defaults; monitor via INFO persistence) --------
# auto-aof-rewrite-percentage 100
# auto-aof-rewrite-min-size 64mb

```

Eviction policy `noeviction` and related behavior reference.[16, 22]

3. **Deploy the service.**
   (a) Push the repo with the files above; Render will build and run the private service.
   (b) Once live, note internal connection: `redis://:pA$$-exAmplE-render@redis-selfhosted:6379`. (For TLS-enabled endpoints, use `rediss://` and `-tls` with `redis-cli`.)[20]
4. **Wire BullMQ to self-hosted Redis (Node.js).**
   (a) In API/Worker environment, set:

```
export REDIS_URL="redis://:pA$$-exAmplE-render@redis-selfhosted:6379"
```

   (b) Example usage:

```
import { Queue } from 'bullmq';
import IORedis from 'ioredis';

const redis = new IORedis(process.env.REDIS_URL!);
export const scoreMatchesQ = new Queue('score-matches', { connection: redis });
```

5. **Backups (Render scheduled job or CI).**
   (a) Option A — *Render Cron Job (separate private service)*: create a small private service that runs `backup-aof.sh` via cron (e.g., s6 or crond in Alpine) at **03:00 UTC**.
   (b) Option B — *CI Scheduled Workflow*: run the backup script from a runner with network access to the private service.

Listing 159: scripts/backup-aof.sh

```
#!/usr/bin/env bash
set -euo pipefail
STAMP=$(date -u +%Y%m%dT%H%M%SZ)
REDIS_URL="${REDIS_URL:-redis://:pA$$-exAmplE-render@redis-selfhosted:6379}"
BACKUP_DIR="${BACKUP_DIR:-/tmp}"
ARCHIVE="${BACKUP_DIR}/redis-aof-${STAMP}.tar.gz"

# Ensure AOF is enabled and fsynced
redis-cli -u "$REDIS_URL" CONFIG GET appendonly | grep -q yes
redis-cli -u "$REDIS_URL" CONFIG GET appendfsync | grep -q everysec

# Optional: trigger BGREWRITEAOF to compact the AOF if not running
AOF_REWRITE=$(redis-cli -u "$REDIS_URL" INFO persistence | grep '^
    aof_rewrite_in_progress:' | cut -d: -f2)
if [ "$AOF_REWRITE" = "0" ]; then
  redis-cli -u "$REDIS_URL" BGREWRITEAOF || true
fi

# Wait a moment to let rewrite settle (best-effort)
sleep 10

# Archive AOF/RDB from the mounted /data (via a one-off tar in a scratch container
    is ideal;
# here we assume this script runs on a host/runner with access to /data via network
    or sidecar)
# If running inside the same pod/container with /data mounted:
tar -czf "$ARCHIVE" -C /data . \
  --exclude='lost+found' \
  --warning=no-file-changed || true

echo "Created archive: ${ARCHIVE}"

# Example: upload to S3-compatible storage (set AWS_* envs)
# aws s3 cp "$ARCHIVE" "s3://my-redis-backups/${STAMP}/redis-aof.tar.gz" --sse
    AES256

```

AOF rewrite command semantics and safety.[18]

6. **Connectivity sanity.**

```
# Expect PONG (no TLS; private network):
redis-cli -u "redis://:pA$$-exAmplE-render@redis-selfhosted:6379" PING
```

For TLS-enabled deployments, `redis-cli` supports `-tls` and CA flags; use `rediss://` URLs.[20]

15.3.3.2.2 Test.

Test maintenance, durability & recovery.

1. **AOF durability smoke.**
   (a) Write a marker: `SET bullmq:test:marker "mvp-2025-09-21T10:00Z"`.
   (b) Wait > 1 s (fsync window), then run: `INFO persistence` and verify `aof_enabled:1`.[17]
   (c) Trigger a **Manual Redeploy** of `redis-selfhosted` in Render (causes container restart).
   (d) After healthy, `GET bullmq:test:marker` should return the value.
2. **AOF rewrite health.**
   (a) Enqueue **1,000** small jobs via BullMQ (or generate write load with a script).
   (b) Check `INFO persistence`: `aof_last_bgrewrite_status:ok`, `aof_rewrite_in_progress:0`.[17]
3. **Eviction policy guard.**
   (a) `CONFIG GET maxmemory-policy` should be `noeviction`.[16, 22]
   (b) If memory is pressured, verify writes fail loudly (no silent drops).
4. **Backup & point-in-time restore drill.**
   (a) Run `backup-aof.sh` and confirm an archive file exists (e.g., `/tmp/redis-aof-<STAMP>.tar.gz`) and uploaded to your bucket.
   (b) **Restore to a staging clone**: create a second private service `redis-restore-drill` with a fresh disk; stop the service; place the archived `appendonly.aof` into `/data`; start the service; Redis will *replay* AOF.[15]
   (c) Connect a staging worker to `redis-restore-drill` and verify the sentinel key and representative BullMQ keys exist (e.g., `bull:score-matches:wait`).
5. **Auth enforcement.**
   (a) `redis-cli -u redis://:wrong@redis-selfhosted:6379 PING` should fail with `NOAUTH`.
   (b) Only a correct password should yield `PONG`.

15.3.3.2.3 Results.

Passed-results variables (verification matrix).

Area	Check / Command	Pass Criteria (Expected)
AOF enabled & policy	`CONFIG GET appendonly`, `CONFIG GET appendfsync`	`appendonly = yes`, `appendfsync = everysec`.
AOF health	`INFO persistence`	`aof_enabled:1`, `aof_last_bgrewrite_status:ok`, `aof_rewrite_in_progress:0`.
Eviction policy	`CONFIG GET maxmemory-policy`	`noeviction`.
Auth	`redis-cli -u redis://:pA$$-exAmplE-render@redis-selfhosted:6379 PING`	`PONG` only when authenticated.
Durability smoke (restart)	Redeploy service, then `GET bullmq:test:marker`	Value persists across container restart.
Backup & restore drill	Restore to `redis-restore-drill` from archive	Sentinel key present; representative BullMQ keys exist; staging worker round-trip succeeds.

15.3.3.2.4 Rationale.

- **AOF every second** → balance durability and throughput for queue metadata; worst case $\leq$ 1s loss while avoiding the latency of `always` fsync.
- **noeviction** → never silently drop BullMQ keys under pressure; fail fast so autoscaling/ops can react (clean up, scale RAM/disk) instead of corrupting queue state.
- **Private network & TLS** → Render Private Service is not Internet-exposed; baseline is auth on a private network. If compliance requires in-transit encryption internally, front Redis with a TLS-terminating proxy and use `rediss://`.
- **Single-instance tradeoff** → unlike managed HA, this setup favors simplicity + fast recovery (container restart + AOF replay). Acceptable for BullMQ metadata when workers are idempotent and jobs are re-drivable.
- **Backups & drills** → nightly AOF archives give point-in-time restore by placing `appendonly.aof` onto a fresh disk and letting Redis replay; verification uses a sentinel key and representative BullMQ keys.
- **AOF rewrite** → leave defaults; monitor `aof_last_bgrewrite_status=ok` and `aof_rewrite_in_progress`; optionally trigger `BGREWRITEAOF` pre-backup to compact artifacts.
- **Operational guardrails** → alerts at 80% memory and disk; track `aof_enabled` and rewrite health; treat evictions/restarts as surfaced failures, not silent data loss paths.

15.3.3.2.5 Monitoring.

- **AOF health:** track `aof_enabled`, `aof_last_bgrewrite_status`, `aof_rewrite_in_progress`.[17]
- **Capacity:** alert at 80% `used_memory` and disk utilization for `/data`.
- **Latency:** observe command latency during load and during AOF rewrites.
- **BullMQ semantics:** design handlers idempotent (*at-least-once* delivery).
- **Disk & AOF rewrite:** alert on `/data` usage $\geq 80\%$ and on `aof_last_bgrewrite_status` $\neq$ `ok` or prolonged `aof_rewrite_in_progress`.

15.3.4 Kill Switches & Circuit Breaking

Control	MVP Target	Notes / Basis
Per-tenant OpenAI breaker	Trip on >20% 5xx over 5 min	Open, then half-open after 60 s; fall back to cached answers. :contentReference[oaicite:13]index=13
Global LLM rate limit	50–100 QPS cap	Budget against provider limits; sheds load safely.
Queue global pause	Admin toggle	Halts consumers to stop churn during incidents.
Read-only mode	Feature flag	Protect DB during recovery/maintenance.

15.3.5 SLO Guardrails & Alerting

Signal	Budget/Threshold	Notes / Basis
API latency	p95 $\leq$ 500 ms	Breach $\Rightarrow$ scale API, throttle producers.
Vector KNN latency	p95 $\leq$ 200 ms	Tune probes/ef_search under load.
Error-budget burn	2-window burn alerts	5m/1h (page) and 30m/6h (ticket) SRE pattern. :contentReference[oaicite:14]index=14
Queue backlog	>10k jobs for 5 min	Auto-scale workers; rate-limit producers.

15.3.6 Deploy Safety & Planned Disruption

Control	MVP Target	Notes / Basis
Rollouts	Rolling/blue-green	Health checks gate step; quick rollback.
Health probes	`/healthz` 10s	Fail fast; remove from LB before kill.
PodDisruptionBudget	minAvailable = 80%	Prevents voluntary disruptions from draining all pods. :contentReference[oaicite:15]index=15
Config deploys	Feature flags	Dark launch; progressive exposure.

16 Conclusion

Status & Intent. This is a *theory-first, non-final* internal design memo. It focuses on the most important architectural choices and interfaces for the Bidion MVP and uses *illustrative* configuration values and targets (e.g., SLOs, sizes, pool limits) to align the team. All numbers, vendor selections, and thresholds are subject to change during implementation and will be refined through spikes, load tests, and RFCs.

This document outlines a practical design for Bidion MVP: a stateless API and worker tier backed by PostgreSQL + `pgvector` with RLS-based multitenancy, queue-driven ingest and recompute, and a measured scaling path (partitions, replicas, HNSW, caching) that preserves cost proportionality.

The Bidion MVP uses a pragmatic, evolvable stack: a **Next.js**/React frontend with SSR/ISR and edge/CDN caching for fast UX and simple CI/CD; a **NestJS** API with **BullMQ** workers for asynchronous pipelines; and **Supabase Postgres** + `pgvector` for retrieval and matching.

On the platform side, we enforce tenant safety with Postgres **RLS**, control cost via batching and caching, and hold SLOs (API p95 ≤ 500 ms; KNN p95 ≤ 200 ms) with **k6**-based checks. The design is intentionally MVP-first: single-region by default, horizontal scale of stateless tiers, and a clear path to partitions and replicas as data or traffic grows. Non-goals for MVP (deferred to v1+) include advanced multi-region data strategies, large-scale full-text+vector hybrids, and automated capacity planning.

The testing and observability plans are specified as part of the design to reduce integration risk later, but remain conceptual until implemented. As an internal specification, this serves as the shared blueprint for initial build-out, early validation with pilot orgs, and controlled evolution toward v1.

17 Acknowledgements

We appreciate contributions and reviews from:

- **Bidion Engineering:** platform & application scaffolding, CI/perf pipelines, and observability setup.
- **Envision MSP stakeholders:** domain guidance on RFP workflows, proposal drafting needs, and success metrics.
- **Community & OSS:** NestJS, pgvector, BullMQ, k6, and related tooling ecosystems that underpin this MVP.

References

[1] Supabase Docs — Point-in-Time Recovery (PITR): overview, RPO characteristics, restore workflow. https://supabase.com/docs/guides/platform/backups#pitr

[2] PostgreSQL Docs — Continuous Archiving & Point-in-Time Recovery (PITR): WAL archiving, base backups, recovery to a timestamp. https://www.postgresql.org/docs/current/continuous-archiving.html

[3] Supabase Docs — Scheduling (server-side) with cron / `pg_cron`. https://supabase.com/docs/guides/database/extensions/pg_cron

[4] `pg_cron` (official) — Simple cron-based job scheduler for Postgres. https://github.com/citusdata/pg_cron

[5] BullMQ Docs — Retries, backoff, and pause/resume support. https://docs.bullmq.io/guide/retries

[6] PostgreSQL Docs — Backup/Recovery and vacuum guidance (see Vacuuming & PITR sections). https://www.postgresql.org/docs/current/routine-vacuuming.html

[7] BullMQ Docs — Overview / API (Queue pause/resume, job options, worker close). https://api.docs.bullmq.io/classes/v4.Queue.html

[8] KEDA Docs — Concepts: External Scalers (scaling from external systems incl. Redis). https://keda.sh/docs/2.17/concepts/external-scalers/

[9] Redis Docs — TLS configuration and secure connections. https://redis.io/docs/latest/operate/oss_and_stack/management/security/encryption/tls/

[10] Redis Enterprise/Cloud — Schedule backups and export/restore workflows. https://redis.io/docs/latest/operate/rs/databases/import-export/schedule-backups/

[11] Redis Cloud/Enterprise — Operate docs hub (creating DBs, HA/replication, features). https://redis.io/docs/latest/operate/

[12] Redis Commands — `LATENCY DOCTOR`. https://redis.io/commands/latency-doctor/

[13] Render Docs — Private Services. `https://render.com/docs/private-services` (accessed 2025-09-22).

[14] Render Docs — Persistent Disks. `https://render.com/docs/disks` (accessed 2025-09-22).

[15] Redis Docs — Persistence (AOF/RDB). `https://redis.io/docs/latest/operate/oss_and_stack/management/persistence/` (accessed 2025-09-22).

[16] Redis Docs — Eviction policies (`maxmemory-policy`, `noeviction`). https://redis.io/docs/latest/develop/reference/eviction/

[17] Redis Command Reference — INFO (persistence section fields). https://redis.io/docs/latest/commands/info/ (accessed 2025-09-22).

[18] Redis Command Reference — BGREWRITEAOF. https://redis.io/docs/latest/commands/bgrewriteaof/ (accessed 2025-09-22).

[19] Redis Docs — TLS (encryption at transport). https://redis.io/docs/latest/operate/oss_and_stack/management/security/encryption/ (accessed 2025-09-22).

[20] Redis Tools — redis-cli (TLS options and usage). https://redis.io/docs/latest/develop/tools/cli/ (accessed 2025-09-22).

[21] Docker Hub — Official Redis Image (7.2-alpine). https://hub.docker.com/_/redis (accessed 2025-09-22).

[22] Redis Enterprise Docs — Eviction Policy Overview. https://redis.io/docs/latest/operate/rs/databases/memory-performance/eviction-policy/ (accessed 2025-09-22).

[23] Supabase Docs — Connect to Postgres (pooled vs direct URLs, Connect panel). https://supabase.com/docs/guides/database/connecting-to-postgres.

[24] Supabase Docs — Connection management (Supavisor pool sizes, direct connections). https://supabase.com/docs/guides/database/connection-management.

[25] Supabase Docs — pgvector: enable the vector extension and usage overview. https://supabase.com/docs/guides/database/extensions/pgvector.

[26] Supabase Docs — Postgres Extensions: enabling/disabling via Dashboard. https://supabase.com/docs/guides/database/extensions.

[27] pgvector (GitHub) — Index types (IVFFLAT, HNSW), operators like <->. https://github.com/pgvector/pgvector.

[28] AWS Database Blog — pgvector indexing deep dive (IVFFLAT vs HNSW). https://aws.amazon.com/blogs/database/optimize-generative-ai-applications-with-pgvector-indexing-a-deep-dive-into-ivfflat-and-hnsw-techniques/.

[29] Supabase Docs — Row Level Security: enabling RLS and writing policies. https://supabase.com/docs/guides/database/postgres/row-level-security.

[30] pganalyze — Storing and querying vector data in Postgres with pgvector. https://pganalyze.com/blog/5mins-postgres-vectors-pgvector.

[31] AWS Docs — General Purpose SSD (gp3/gp2) baseline IOPS scale with size. https://docs.aws.amazon.com/ebs/latest/userguide/general-purpose.html.

[32] AWS Docs — EBS I/O characteristics (throughput limits can cap effective IOPS). https://docs.aws.amazon.com/ebs/latest/userguide/ebs-io-characteristics.html.

[33] Supabase Docs — PITR restore time depends on WAL volume and last full backup. https://supabase.com/docs/guides/troubleshooting/how-long-does-it-take-to-restore-a-database-from-a-point-in-time-backup-pitr-qO8gOG.

[34] Supabase Docs — JavaScript: Create a bucket (Storage API reference). https://docs-supabase.vercel.app/docs/reference/javascript/storage-createbucket

[35] Supabase Docs — JavaScript: Create a signed upload URL (Storage API reference). https://docs-supabase.vercel.app/docs/reference/javascript/storage-from-createsigneduploadurl

[36] Supabase Docs — JavaScript: Upload to a signed URL (Storage API reference). https://docs-supabase.vercel.app/docs/reference/javascript/storage-from-uploadsignedurl

[37] Supabase Docs — Storage: Access control (policies, public vs private, signed URLs). https://docs-supabase.vercel.app/docs/guides/storage/access-control

[38] GitHub — openai/openai-node (Official JS/TS library; environment usage and examples). https://github.com/openai/openai-node.

[39] OpenAI Platform Docs — API Reference: Embeddings (endpoint, parameters, response shape). https://platform.openai.com/docs/api-reference/embeddings.

[40] OpenAI Platform Docs — Rate limits (limits, headers, best practices). https://platform.openai.com/docs/guides/rate-limits.

[41] OpenAI Platform Docs — Error handling and retries (backoff guidance). https://platform.openai.com/docs/guides/retries.

[42] OpenAI API Docs — Authentication (Manage API keys in Organization settings; Bearer usage). https://platform.openai.com/docs/api-reference/authentication

[43] OpenAI Help Center — Assign API key permissions in the API Platform. https://help.openai.com/en/articles/9520199-assign-api-key-permissions

[44] OpenAI Help Center — Enabling or disabling Multi-Factor Authentication (MFA). https://help.openai.com/en/articles/7967234-enabling-or-disabling-multi-factor-authentication-mfa