Vishwaas.ai

Technical Architecture Brief

Overview

Vishwaas AI is a cloud-native, multi-tenant SaaS platform built on a modern TypeScript-first stack. It delivers DPDP Act compliance capabilities through five architectural pillars: strict multi-tenancy, non-repudiable consent, event-driven propagation, identity unification, and defence-in-depth security. The entire platform runs in AWS Mumbai (ap-south-1) to satisfy India data residency requirements.

Technology Stack

Frontend — Next.js 14 (App Router)

Layer	Technology	Purpose
Framework	Next.js 14 App Router	SSR, RSC, file-based routing, API routes
Language	TypeScript 5.x (strict mode)	Type safety throughout
Styling	Tailwind CSS 3.x + shadcn/ui	Accessible, Radix-based component library
State	Zustand 4.x + TanStack Query v5	Client state + server-state caching and sync
Forms	React Hook Form + Zod	Schema-validated form handling
i18n	next-intl	22 Indian languages + English
Charts	Recharts 2.x	Dashboard analytics visualisation
Rich Text	TipTap 2.x	Privacy notice authoring
PDF	@react-pdf/renderer	Consent receipts, audit reports
HTTP	Axios	JWT + tenant header interceptors

Routing pattern: Every admin and portal page exists in two forms — a base path (/admin/...) used by super_admin, and a tenant-scoped path (/[tenantSlug]/admin/...) used by all tenant roles. Both resolve to the same page components.

Backend — NestJS 10 (API)

Layer	Technology	Purpose
Runtime	Node.js 20 LTS	SSR, RSC, file-based routing, API routes
Framework	NestJS 10	Type safety throughout
Language	TypeScript 5.x (strict mode)	Accessible, Radix-based component library
ORM	Prisma 5.x	Client state + server-state caching and sync
Auth	Passport.js + @nestjs/jwt	Schema-validated form handling
Authorisation	CASL 6.x	22 Indian languages + English
API Docs	@nestjs/swagger (OpenAPI 3.0)	Dashboard analytics visualisation
Job Queue	BullMQ (Redis-backed)	Privacy notice authoring
Email	Nodemailer	Consent receipts, audit reports
Validation	class-validator + class-transformer	JWT + tenant header interceptors

Data Layer

Store	Technology	Role
Primary DB	PostgreSQL 16	ACID transactions, JSONB, Row-Level Security, UUID v7 PKs
Audit Ledger	ostgreSQL audit schema	Append-only hash-chained event log; INSERT + SELECT only at DB role level
Time-Series	TimescaleDB (PG extension)	Dashboard trend analytics (consent grant/withdrawal rates)
Cache / OTP	Redis 7.x (ioredis)	OTP codes (DB 0), sessions, consent status cache (DB 2); isolated DBs prevent cross-feature pollution
Search	Elasticsearch 8.x	Audit log search, consent record full-text search
Message Bus	Kafka (KafkaJS, KRaft mode)	Consent events, rights events, breach events, webhook dispatch, notification dispatch
File Store	AWS S3 / MinIO	Privacy notices, DPIAs, consent PDFs, CSV uploads; SSE encryption; ap-south-1

Multi-Tenancy Architecture

Vishwaas AI is a shared-infrastructure, strict-isolation multi-tenant system. All customer data lives in a single PostgreSQL instance but is isolated by three complementary mechanisms:

Request → JWT (carries tenant_id)
→ TenantContextMiddleware (extracts tenant_id)
→ Prisma extension (auto-injects WHERE tenant_id = ?)
→ PostgreSQL Row-Level Security (enforces tenant_id at DB level)

Isolation layers

Layer	Mechanism	What It Prevents
API	TenantContextMiddleware injects tenant_id into every request context	Cross-tenant queries from reaching the ORM
ORM	Prisma client extension wraps all queries with AND tenant_id = :id	Accidental omission of tenant filter
Database	PostgreSQL Row-Level Security on all app schema tables	Cross-tenant data leakage even via direct SQL
Auth	super_admin JWT has no tenant_id; consent/DPR/notice APIs reject it with 403	Super admins accessing customer data

Every table in the app schema carries a non-nullable tenant_id UUID column. UUID v7 primary keys provide time-ordered uniqueness without a central sequence.

Tenant routing: Each organisation receives a tenant slug (e.g. acme-corp). Admin users log in at /{slug}/login; data principals at /{slug}/portal/login. The slug is embedded in the JWT and validated on every API call.

Non-Repudiable Consent - The Hash Chain

The consent ledger is the legal heart of Vishwaas AI. Every consent record is cryptographically bound to its neighbours, making retroactive tampering mathematically detectable.

Record N-1

data_principal_id
purpose_code
action (GRANTED)
consent_text_snap
ip_address
record_hash [H(N-1)]
chain_hash [C(N-1)]
digital_sig
tsa_token

Record N

data_principal_id
purpose_code
action (WITHDRAWN)
consent_text_snap
ip_address
record_hash [H(N-1)]
chain_hash [C(N-1] = H(H(N) + C(N-1)))
digital_sig [RSA(H(N), priv_key)]
tsa_token [RFC 3161]

Four-layer proof per record

Proof	Algorithm	What It Proves
record_hash	SHA-256 of deterministic JSON	Record content was not modified
chain_hash	SHA-256(record_hash + prev_chain_hash)	Record sequence was not altered; no record was inserted or deleted
digital_signature	RSA-2048 / SHA-256 (AWS KMS)	Record was created by this organisation's authorised system
tsa_token	RFC 3161 Timestamp Authority	The exact moment of consent is legally defensible

Genesis anchor: The first record in each tenant's chain uses SHA-256("VISHWAAS_AI_GENESIS_" + tenant_id) as the previous chain hash — cryptographically binding the chain to the tenant.

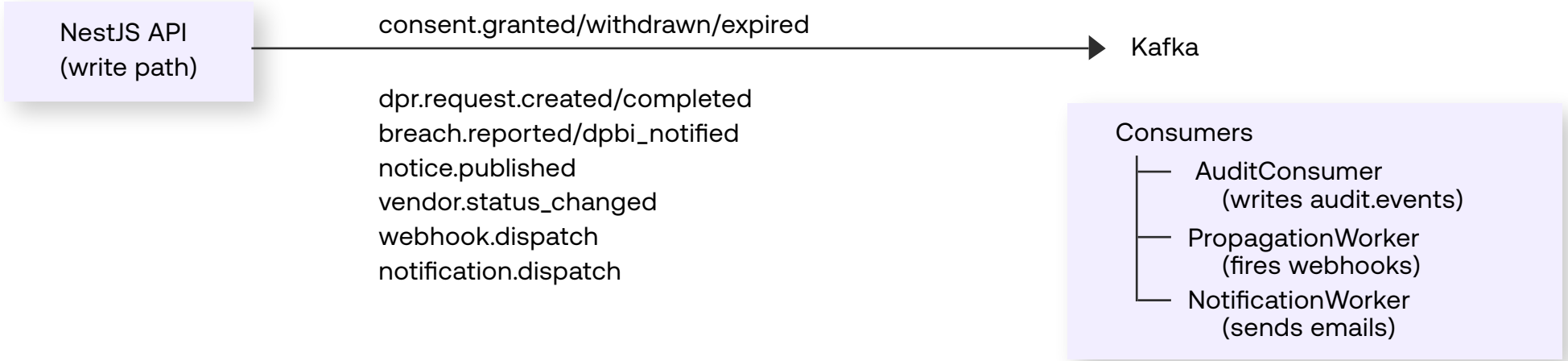
Append-only enforcement: The consent_records table and audit.events table have UPDATE and DELETE revoked at the PostgreSQL role level:

```
REVOKE UPDATE, DELETE ON consent_records FROM crosstrust_app;
REVOKE UPDATE, DELETE ON audit.events    FROM crosstrust_app;
```

This is not an application-layer guard — it is a database-level permission enforced by PostgreSQL itself.

Event-Driven Architecture

All compliance-significant mutations produce Kafka events. This decouples the API from downstream processing and ensures audit, propagation, and notification are reliable and independently scalable.



Kafka topics

Topic	Producers	Consumers
consent.events	Consent API	AuditConsumer, PropagationWorker, CacheInvalidator
dpr.events	DPR API	AuditConsumer, NotificationWorker
breach.events	Breach API	AuditConsumer, NotificationWorker
notice.events	Notice API	AuditConsumer
audit.events	All modules	AuditConsumer (hash-chain writer)
webhook.dispatch	PropagationService	WebhookDeliveryWorker (HMAC delivery + retry)
notification.dispatch	All modules	EmailWorker (Nodemailer / SES)

Consent Propagation flow:

Consent change (portal toggle or API call)

→ consent.events published to Kafka

→ PropagationWorker reads event

→ Queries PropagationWebhook registry (per tenant, per downstream app)

→ HTTP POST to each endpoint with X-VishwaasAI-Signature header

→ PropagationDeliveryLog updated (HTTP status, latency)

→ On failure: exponential backoff (1s → 2s → 4s → 8s → 16s → 30s)

→ After max retries: dead-letter queue

→ Redis consent-status cache invalidated

Target SLA: Consent change → webhook confirmed delivery in < 5 seconds.

Identity Unification Pipeline

Vishwaas AI resolves fragmented consumer identities across source systems into a single canonical data_principals record:

Source System (Salesforce / HRIS / CSV)
→ Ingestion Service (field normalisation: email, phone, name, PAN, Aadhaar)
→ staging_records table (raw + normalised fields, batch_id)
→ Identity Matchers:
Deterministic → exact email / PAN / Aadhaar match → auto-link (100% confidence)
Probabilistic → Jaro-Winkler name + DOB + city → review queue (≥ 85%)
→ Human Review Queue (approve / reject / defer)
→ external_identity_map (staging_record_id → data_principal_id)
→ merge_audit_trail (append-only merge decision log)
→ canonical data_principals (source_system_count++, unification_status updated)

The result is one authoritative consent record per individual — not one per system — ensuring DPR requests and erasure jobs cover every data silo.

Security Architecture

Authentication — Passwordless OTP Only

ser enters email

→ crypto.randomInt(100000, 999999) generates OTP

→ Stored in Redis: key otp:{SHA-256(email)}, TTL 600s

→ Delivered via Nodemailer (MailHog dev / AWS SES prod)

→ User submits OTP → verified against Redis

→ Issues: JWT access token (15m) + refresh token (7d)

→ OTP key deleted from Redis (single-use)

Rate limits: 3 OTP requests / 5 min per email · 5 verify attempts per OTP · automatic lockout.

Authentication — Passwordless OTP Only

Scope	Algorithm	Key Management
PII at rest (email, phone, name)	AES-256-GCM	App-layer; key in AWS Secrets Manager
Sensitive PII (Aadhaar, health data)	AES-256-GCM	AWS KMS (envelope encryption)
S3 objects	SSE-S3 / SSE-KMS	per-bucket policy
Transit	TLS 1.3	All API and web traffic
Webhook secrets	AES-256-GCM	Encrypted at rest; revealed once at registration
Consent signing keys	RSA-2048	AWS KMS; per-tenant key pair

PII storage pattern: Plaintext is never persisted. Each PII field is stored as two columns: - email_encrypted BYTEA — AES-256-GCM ciphertext - email_hash VARCHAR(64) — SHA-256 hex for indexed lookups (search without decryption)

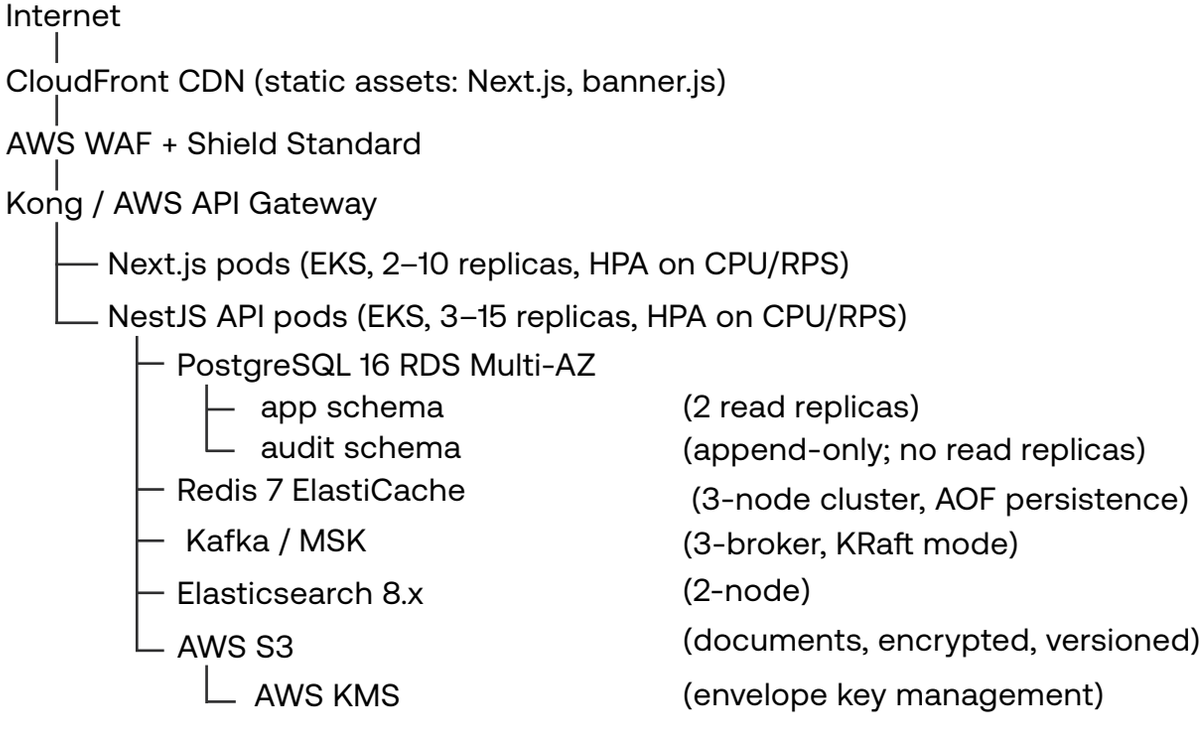
Authorisation — CASL ABAC

JWT → JwtAuthGuard (verify signature + expiry)
→ TenantContextMiddleware (extract tenant_id)
→ PermissionsGuard (CASL ability check)
ability.can(action, resource, conditions)
conditions: { tenant_id, dept_id, own_records_only, ... }
→ Handler executes

11 roles × 15 resource types × 6 action types (create / read / update / delete / approve / export). Role permissions stored in role_permissions with a conditions JSONB column enabling scoped access (e.g. dept_manager reads only their department's consent records).

Deployment Topology

Region: ap-south-1 (Mumbai) — India data residency



CI/CD: GitHub Actions → Docker build (multi-stage, distroless prod image) → ECR push → ArgoCD GitOps deploy to EKS.

Observability: Prometheus + Grafana (metrics) · Loki (log aggregation) · OpenTelemetry + Jaeger (distributed tracing) · CloudWatch (AWS resource metrics).

Health endpoints:

```
GET /api/v1/health        → { status, db, redis, kafka, smtp }
GET /api/v1/health/ready  → Kubernetes readiness probe
```

Key Architecture Decisions

Decision	Choice	Rationale
Monorepo	pnpm workspaces + Turborepo	Shared types between frontend and backend with zero drift
ORM	Prisma 5	Type-safe queries, migration tooling, Prisma Studio for visual DB inspection
Auth	Passwordless OTP only	Eliminates credential theft; no password reset flows; no MFA complexity
Audit ledger	Separate PostgreSQL schema with DB-level revoke	Application-layer guards can be bypassed; DB-level REVOKE cannot
Consent hashing	SHA-256 Merkle chain + RSA signatures	Provides legally defensible, independently verifiable non-repudiation
Timestamps	RFC 3161 TSA	Provides court-admissible proof of the moment consent was recorded
Multi-tenancy	Shared DB + RLS (not schema-per-tenant)	Operationally simpler migration management; cost-efficient at scale
Event bus	Kafka (KRaft, no Zookeeper)	Durable, ordered event delivery; decouples audit, propagation, and notifications
Cache isolation	Redis DB 0 (OTP), DB 2 (consent cache)	Prevents accidental cross-feature cache poisoning
PII storage	Encrypt + store hash separately	Enables indexed lookups (DPR search) without ever storing plaintext
India residency	AWS ap-south-1 exclusively	DPDP Act data localisation; no cross-region replication of personal data