Software Architecture -- SOC Audit Documentation

Generated: 2026-02-11 **Scope:** Full-system software architecture review covering `/app`, `/admin`, and `/devops` **Document Type:** SOC 2 Type II -- System Description **Classification:** Internal -- Auditor Distribution

---

1. Executive Summary

The system is a full-stack TypeScript monorepo comprising three sub-projects: a main application (`/app`), a DevOps administration portal (`/admin`), and Terraform infrastructure provisioning (`/devops`). The main application uses a React + Vite client with a NestJS server backed by PostgreSQL via Prisma ORM. The architecture follows a modular design with a clear separation between **core** modules (authentication, RBAC, MFA, encryption, audit) and **app** modules (business-specific features including meeting management, AI-driven transcript processing, project tracking, and stakeholder analysis). Cross-cutting concerns -- event-driven communication, field-level encryption, structured logging, and role-based access control -- are implemented as framework-level middleware and guards.

The admin portal is an independent full-stack application providing infrastructure management, security scanning, database migration, S3 backup management, and interactive SSH terminals, secured via Auth0 JWT authentication with AWS STS credential exchange.

The infrastructure layer uses Terraform for multi-cloud provisioning with Auth0 OIDC integration for AWS IAM role assumption.

The database schema contains 35 tables and 18 enums, covering authentication, MFA, RBAC, audit logging, encryption management, meeting processing, project management, stakeholder tracking, and system infrastructure.

---

2. Monorepo Structure

```
/
+-- app/                        # Main application
|   +-- client/                 # React + Vite frontend (port 5174 dev)
|   +-- server/                 # NestJS backend (port 3000)
|       +-- src/
|       |   +-- modules/
|       |   |   +-- core/       # 22 system-level modules
|       |   |   +-- app/        # 7 business modules
|       |   +-- config/
|       |   |   +-- core/       # 15 typed configuration files
|       |   |   +-- app/        # 1 app-specific config
|       |   +-- common/         # Events, exceptions, utilities
|       +-- prisma/
|           +-- schema.prisma   # 35 tables, 18 enums
+-- admin/                      # DevOps administration portal
|   +-- client/                 # React + Vite frontend (port 4100)
```

```
|   +-- server/                # NestJS backend (port 4000)
|       +-- src/modules/       # 12 admin modules
+-- devops/                    # Terraform infrastructure
+-- CLAUDE.md                  # Root project guidance
```

Sub-project isolation: The main application (`/app`) and admin portal (`/admin`) run as completely separate NestJS processes on different ports with independent authentication mechanisms (JWT with local credentials vs. Auth0). They share no runtime state or database connections.

---

3. Server Architecture (NestJS)

3.1 Module Organization

Modules are separated into **core** (system-level, reusable across applications) and **app** (business-specific to this deployment):

Core Modules (`app/server/src/modules/core/`)

MODULE	PURPOSE	KEY PATTERNS
`auth/`	Login, registration, JWT, password reset, OAuth (Google/Microsoft/LDAP)	Passport strategies, httpOnly cookie tokens, session management, token blacklist
`rbac/`	Roles, permissions, organizations, Casbin policies	Priority-based hierarchy (SuperAdmin > Admin > Consultant > Customer > Unapproved), organization-scoped JWT
`mfa/`	TOTP setup, backup codes, email OTP, trusted devices, grace periods	Speakeasy TOTP, AES-256-GCM encryption of secrets, configurable mandatory roles
`audit/`	Event logging for compliance	Event-driven listener, 31 audit event types, structured metadata
`alerting/`	Alerts, notifications, digest emails	Audit event subscription, severity mapping (LOW/MEDIUM/HIGH/CRITICAL), digest scheduling
`email/`	SMTP email sending	Template-based, configurable SMTP host/port/credentials
`encryption/`	AES-256 field encryption via Prisma extension middleware	Versioned keys (pii/mfa/hipaa), deterministic and non-deterministic modes
`invitation/`	User invitations to organizations	Token-based, role-scoped, expiration tracking
`settings/`	Persistent system configuration	Key-value JSON store with typed retrieval

MODULE	PURPOSE	KEY PATTERNS
`system/`	Backup, restore, reset operations	Email verification required, SQL dump/restore, file upload
`instance-health/`	CPU, RAM, disk monitoring	Per-instance tracking with heartbeat, leader election flag
`database-health/`	DB performance monitoring	Connection pool stats, query performance
`key-rotation/`	Encryption key rotation	Batch migration with pause/resume, per-record tracking, error logging
`retention/`	Data cleanup after retention period	Configurable retention (audit: 365d, security audit: 730d, soft-delete: 90d)
`scheduler/`	Leader election, distributed locks	PostgreSQL advisory locks for single-instance job execution
`logger/`	Structured application logging	Winston, SIEM integration, HTTP request/response interceptor
`prisma/`	Database ORM service	Connection pooling, encryption extension
`ai/`	LLM integration	Token tracking, cost calculation, latency logging
`background-jobs/`	Asynchronous job processing	Status tracking (PENDING/RUNNING/COMPLETED/FAILED), event-driven execution
`health/`	Application health checks	Startup/readiness probes
`documentation/`	System documentation endpoints	Runtime documentation serving
`user-context/`	User context resolution	Request-scoped user resolution

App Modules (`app/server/src/modules/app/`)

MODULE	PURPOSE	KEY PATTERNS
`meetings/`	Meeting sync from Fireflies, transcript processing, customer access	External API integration, background job AI processing, customer report approval workflow
`templates/`	AI prompt templates for meeting and brief processing	CRUD with type distinction (MEETING/BRIEF), globally scoped
`pdf/`	PDF generation from meeting output and briefs	Binary storage in database

MODULE	PURPOSE	KEY PATTERNS
`projects/`	Project management with organization scoping	Status lifecycle (DISCOVERY > ACTIVE > ON_HOLD > COMPLETED > ARCHIVED)
`actions/`	Decision and action item tracking	Priority levels, flagging, meeting linkage, self-referencing relations
`stakeholders/`	Stakeholder management and insight extraction	Type classification (INTERNAL/EXTERNAL/UNKNOWN), per-meeting insights
`briefs/`	Multi-meeting AI-generated briefs	Date range or specific meeting selection, notification tracking

3.2 Global Modules

Only two modules use `@Global()`:

- **PrismaModule** -- Database access available everywhere without explicit import
- **LoggerModule** -- Logging available everywhere without explicit import

All other modules must be explicitly imported where needed, preventing hidden dependencies.

3.3 Module Patterns

Controller -> Service -> Repository:

```
Controller (thin, DTO validation)
  -> Service (business logic, event emission)
    -> Repository (Prisma database access)
```

Key Conventions:

- Controllers never return Prisma entities directly; always map to response DTOs via `static fromEntity()`
- Repositories define interfaces (e.g., `IAuthRepository`) for testability and implementation swapping
- Cross-module communication uses `@nestjs/event-emitter` domain events rather than direct service imports
- Configuration uses NestJS `registerAs()` pattern with typed, startup-validated config
- Circular dependencies must not be resolved using `forwardRef()`
- Barrel files (index.ts re-exports) are not used in server modules

3.4 Startup Security Validation

The server enforces mandatory security checks at bootstrap:

- **JWT_SECRET** must be defined or the process exits immediately (`process.exit(1)`)

- **ENCRYPTION_KEYS_FILE** must point to a valid JSON file with correct structure or startup fails
- Encryption key file is validated for: three required key types (pii, mfa, hipaa), valid version patterns, 64-character hex keys, unique key IDs across all types
- Helmet middleware is applied with strict Content-Security-Policy directives
- Global `ValidationPipe` with `whitelist: true` and `forbidNonWhitelisted: true` rejects unknown fields
- Swagger UI is disabled in production (`NODE_ENV === 'production'`)

3.5 Error Handling

A base `AppException` class extends `HttpException` with consistent structure:

```
interface AppExceptionResponse {
  code: string          // Machine-readable error code (e.g., "ENTITY_NOT_FOUND")
  message: string       // Human-readable description
  statusCode: number    // HTTP status
  timestamp: string     // ISO 8601
  metadata?: object     // Additional context (entity name, field errors)
}
```

Specialized exception types:

EXCEPTION CLASS	HTTP STATUS	CODE
`EntityNotFoundException`	404	`ENTITY_NOT_FOUND`
`ValidationException`	400	`VALIDATION_ERROR`
`UnauthorizedException`	401	`UNAUTHORIZED`
`ForbiddenException`	403	`FORBIDDEN`
`ConflictException`	409	`CONFLICT`
`InternalServerException`	500	`INTERNAL_ERROR`

A global `AppExceptionFilter` catches all exceptions, normalizes the response format, and prevents stack trace leakage in production. Unhandled exceptions are logged with full stack traces server-side while returning only `"Internal server error"` to the client.

3.6 Guards and Middleware Pipeline

Request processing order:

1. **Helmet** -- HTTP security headers including Content-Security-Policy
2. **Cookie Parser** -- Parse httpOnly cookies for JWT extraction
3. **Global HTTP Interceptor** -- Logs method, URL, status, duration, userId, IP (with body sanitization)

4. **ThrottlerGuard** -- Global rate limiting (50 requests/minute per IP, Redis-backed when available)
5. **JwtAuthGuard** -- JWT validation from httpOnly cookies (checks `@Public()` decorator to skip)
6. **LoginThrottlerGuard** -- Additional throttling for login endpoint with lockout support
7. **RbacGuard** -- Role-based access (checks `@RequireRole()` decorator against JWT role priority)
8. **ValidationPipe** -- DTO whitelist validation with `forbidNonWhitelisted`
9. **Controller method** -- Business logic execution
10. **Audit event emission** -- Security-relevant events logged asynchronously via event emitter

Available auth guards:

GUARD	PURPOSE
`JwtAuthGuard`	Validates JWT from httpOnly cookies
`GoogleAuthGuard`	Initiates/handles Google OAuth flow
`MicrosoftAuthGuard`	Initiates/handles Microsoft OAuth flow
`LoginThrottlerGuard`	Login-specific rate limiting with account lockout
`RbacGuard`	Role hierarchy enforcement via `@RequireRole()` decorator
`LocalAuthGuard`	Username/password validation

---

4. Client Architecture (React + Vite)

4.1 Component Organization

Components follow the core/app split pattern:

Core Components (`app/client/src/components/core/`):

FOLDER	COMPONENTS	PURPOSE
`auth/`	LoginPage, RegisterPage, ForgotPasswordPage, ResetPasswordPage, UnapprovedPage, AuthProvider	Authentication flows and route protection
`mfa/`	MfaSetupWizard, MfaVerificationModal, TrustedDevicesManager, BackupCodesDisplay, MandatoryMfaSetup, MfaSettingsSection	MFA enrollment, verification, and device management
`settings/`	SettingsModal, UsersSettingsForm, SmtpSettingsForm, AuditSettingsForm, AlertingSettingsForm, OAuthSettingsForm, DatabaseSettingsForm, ClusterSettingsForm, PoliciesSettingsForm, SystemSettingsForm, AiSettingsForm, LogSettingsForm, OrganizationsSettingsForm,	System administration and configuration

FOLDER	COMPONENTS	PURPOSE
	NotificationPreferencesForm, SessionManager, SystemResetButton, InviteUserModal, UserEditModal, DeleteUserModal, AuditLogDetailModal, AlertLogModal, AiLogModal, LogViewerModal, AppSettingsModal	
`background-jobs/`	BackgroundJobsIndicator	Background job status display
`documentation/`	SystemDocsPage	System documentation viewer

App Components (`app/client/src/components/app/`):

FOLDER	COMPONENTS	PURPOSE
`meeting/`	MeetingList, MeetingViewModal, AddMeetingModal, RunTemplatesModal, NotifyModal, AiOutputPickerModal, AiOutputViewerModal, OutputPickerList, PdfPickerModal, CustomerReportsPage, CustomerAccessModal, ApprovalModal	Meeting management, AI processing, PDF generation, customer access control
`template/`	TemplateList, TemplateFormModal	AI prompt template management
`project/`	ProjectList, ProjectDetailPage, ProjectFormModal, ProjectEditModal, ProjectInfoPopup, ProjectActionsTab, ProjectBrainTab, ProjectStakeholdersTab, ProjectBriefsTab, ActionEditModal, StakeholderDetailModal, StakeholderMergePreviewModal, StakeholderPickerModal, StakeholderProfileBlock	Project management with tabbed detail view
`brief/`	GenerateBriefModal, BriefViewModal, BriefNotifyModal	Multi-meeting brief generation and distribution

Shared/Sub Components (`app/client/src/components/sub/`):

Reusable sub-components shared across core and app component boundaries.

Root Components:

COMPONENT	PURPOSE
`Home.tsx`	Main dashboard/landing page
`Navbar.tsx`	Top navigation bar
`OrgSwitcher.tsx`	Organization context switcher

4.2 State Management (Zustand)

Core Stores (`app/client/src/store/core/`):

STORE	PURPOSE
`auth/auth.store.ts`	Auth state, login, MFA flows, organization context, token refresh
`settings/settings.store.ts`	System settings CRUD
`alerting/alerting.store.ts`	Alert management and notification preferences
`ai/ai.store.ts`	AI provider settings and log viewing
`database-health/database-health.store.ts`	Database performance metrics
`instance-health/instance-health.store.ts`	Instance monitoring data
`logs/logs.store.ts`	Application log viewing
`background-jobs/background-jobs.store.ts`	Background job status tracking

App Stores (`app/client/src/store/app/`):

STORE	PURPOSE
`meeting/meeting.store.ts`	Meetings CRUD, AI processing, PDF operations
`meeting/customer-reports.store.ts`	Customer report access and approval
`template/template.store.ts`	Template CRUD
`project/project.store.ts`	Project CRUD, stakeholder management
`brief/brief.store.ts`	Brief generation and management

4.3 API Layer

- **`client.tsx`** -- API wrapper with JWT refresh, automatic token renewal on 401, httpOnly cookie authentication
- **`Api.ts`** -- Auto-generated from `swagger.yaml` (OpenAPI spec) via code generation
- **Proxy:** Vite dev server proxies `/api/*` requests to NestJS backend at `http://localhost:3000`
- **OpenAPI regeneration:** Required when controller endpoints or DTO structures change (`npm run openapi --workspace=server`)

---

5. Database Schema Overview

5.1 ORM and Schema

- **ORM:** Prisma with PostgreSQL provider
- **Schema location:** `app/server/prisma/schema.prisma` (35 tables, 18 enums)
- **DTO Generation:** `prisma-generator-nestjs-dto` auto-generates Create/Update DTOs with class-validator decorators
- **Connection Pooling:** Configurable pool (default min: 2, max: 20, idle timeout: 30s, connection timeout: 5s) with TCP keepAlive

5.2 Table Inventory

Authentication and Sessions (8 tables):

TABLE	PURPOSE	KEY FIELDS
`users`	User accounts	`email` (unique, encrypted), `name`, `isActive`, `failedLoginAttempts`, `lockedUntil`, `tokenInvalidatedAt`
`user_credentials`	Password hashes	`userId` (unique), `passwordHash`
`user_auth_providers`	OAuth provider links	`provider`, `providerId`, `email`, `metadata`
`refresh_tokens`	Session tokens	`token` (unique), `expiresAt`, `revokedAt`, `lastUsedAt`, `userAgent`, `ipAddress`, `deviceName`
`revoked_tokens`	JWT blacklist	`jti` (unique), `expiresAt`, `reason`
`password_reset_codes`	Reset verification	`email` (unique), `code`, `expiresAt`, `attempts`
`password_reset_rate_limits`	Reset throttling	`email` (unique), `count`, `resetAt`
`password_history`	Password reuse prevention	`userId`, `passwordHash`, `createdAt`

Multi-Factor Authentication (5 tables):

TABLE	PURPOSE	KEY FIELDS
`user_mfa_configs`	MFA settings per user	`mfaEnabled`, `preferredMethod`, `totpSecret` (encrypted), `mfaGracePeriodEnd`
`mfa_backup_codes`	One-time recovery codes	`codeHash`, `usedAt`

TABLE	PURPOSE	KEY FIELDS
`mfa_challenges`	Active verification challenges	`challengeType`, `code`, `expiresAt`, `attempts`, `lockedUntil`
`trusted_devices`	Remembered devices	`deviceHash`, `deviceName`, `trustExpiresAt`
`mfa_recovery_requests`	Account recovery	`recoveryEmail`, `tokenHash`, `expiresAt`

RBAC and Multi-Tenancy (5 tables):

TABLE	PURPOSE	KEY FIELDS
`roles`	Role definitions	`name` (unique), `priority` (SuperAdmin=200, Admin=100, Consultant=75, Customer=50, Unapproved=0)
`organizations`	Tenant entities	`name`, `slug` (unique), `isActive`
`user_organization_roles`	User-org-role junction	`userId` + `organizationId` (unique), `roleId`, `isDefault`
`casbin_rules`	Casbin policy storage	`ptype`, `v0` - `v5` (role hierarchy rules)
`user_invitations`	Organization invitations	`email`, `token` (unique), `status` (PENDING/ACCEPTED/EXPIRED/CANCELLED), `expiresAt`

Audit and Compliance (4 tables):

TABLE	PURPOSE	KEY FIELDS
`audit_logs`	Security event log	`eventType` (31 types), `userId`, `organizationId`, `ipAddress`, `userAgent`, `metadata`
`alerts`	Alert notifications	`eventType`, `severity`, `status` (PENDING/SENT/ACKNOWLEDGED), `acknowledgedBy`
`alert_digest_logs`	Digest email tracking	`userId`, `alertCount`, `alertIds`, `emailStatus`
`user_notification_preferences`	Per-user alert config	`receiveAlertDigests`, `digestIntervalMins`, `minSeverity`

Encryption Management (2 tables):

TABLE	PURPOSE	KEY FIELDS
`key_rotation_events`	Key rotation tracking	`keyType`, `fromVersion`, `toVersion`, `status`, `recordsTotal`, `recordsMigrated`, `recordsFailed`
`encryption_key_audits`	Encryption field audits	`keyType`, `keyVersion`, `tableName`, `fieldName`, `recordCount`

Application -- Meetings and Processing (4 tables):

TABLE	PURPOSE	KEY FIELDS
`meetings`	Meeting records	`firefliesId`, `ingestionType` (FIREFLIES/MANUAL), `title`, `transcriptContent`, `rawData` (JSONB), `organisationId`, `projectId`
`meeting_outputs`	AI-processed outputs	`meetingId` + `templateId` (unique), `outputText`, `pdfData` (binary)
`meeting_customer_access`	Customer report permissions	`meetingId` + `templateId` + `userId` (unique), `approvalRequired`, `approvalStatus`
`templates`	AI prompt templates	`name` (unique, global), `prompt`, `type` (MEETING/BRIEF)

Application -- Projects and Stakeholders (4 tables):

TABLE	PURPOSE	KEY FIELDS
`projects`	Project tracking	`name` + `organisationId` (unique), `status`, `startDate`, `endDate`, `createdById`
`actions`	Decisions and action items	`item`, `type` (DECISION/ACTION), `status`, `priority`, `meetingId`, `projectId`, `organisationId`
`stakeholders`	Project stakeholders	`name` + `projectId` (unique), `type` (INTERNAL/EXTERNAL/UNKNOWN), `summary`
`stakeholder_insights`	Per-meeting stakeholder data	`type` (QUESTION/CONCERN/STATEMENT), `content`, `context`

Application -- Briefs (1 table):

TABLE	PURPOSE	KEY FIELDS
`briefs`	Multi-meeting AI briefs	`templateId`, `projectId`, `generatedById`, `dateFrom`, `dateTo`, `meetingIds` (JSONB), `pdfData`

System and Infrastructure (4 tables):

TABLE	PURPOSE	KEY FIELDS
`settings`	System configuration	`name` (unique), `value` (JSONB)
`instance_health`	Instance monitoring	`instanceId` (unique), `hostname`, `isLeader`, CPU/RAM/disk metrics, `lastHeartbeat`
`background_jobs`	Async job tracking	`jobType`, `entityId`, `status`, `statusMessage`, `errorSummary`, `metadata` (JSONB)
`ai_logs`	AI usage tracking	`provider`, `model`, `inputTokens`, `outputTokens`, `costUsd`, `latencyMs`

System Verification (1 table):

TABLE	PURPOSE	KEY FIELDS
`system_verification_codes`	Reset/restore verification	`userId` + `verificationType` (unique), `code`, `expiresAt`

5.3 Key Relationships

- Users belong to multiple organizations via `user_organization_roles` junction table
- Each user-organization assignment has exactly one role
- Meetings are organization-scoped and optionally project-scoped
- Actions link to both a meeting and a project, creating cross-entity traceability
- Stakeholders are project-scoped; stakeholder insights link to both stakeholder and meeting
- Meeting outputs are uniquely constrained by meeting + template pair
- Customer access controls are uniquely constrained by meeting + template + user triple
- Templates are globally scoped (not organization-scoped)
- Briefs aggregate multiple meetings via JSONB array of meeting UUIDs

5.4 Data Conventions

- All tables include `createdAt` and `updatedAt` timestamps
- Soft delete via `isDeleted` boolean flag (preserves audit trail)
- UUIDs (`@db.Uuid`) for all primary keys
- Organization scoping on app-level data via `organisationId` foreign key
- Core data (templates, roles, settings) is globally scoped
- Field-level encryption on PII via Prisma extension middleware
- JSONB columns used for flexible metadata storage
- Database table names use snake_case via `@@map()`
- Column names use snake_case via `@map()`, Prisma fields use camelCase
- Comprehensive indexing on foreign keys, status fields, and commonly queried columns

5.5 Enums

ENUM	VALUES	USED BY
`ActionPriority`	LOW, MEDIUM, HIGH, HIGHEST	Actions
`ActionStatus`	OPEN, IN_PROGRESS, CLOSED_DONE, CLOSED_LOST	Actions
`ActionType`	DECISION, ACTION	Actions
`AlertSeverity`	LOW, MEDIUM, HIGH, CRITICAL	Alerts, Notification Preferences
`AlertStatus`	PENDING, SENT, ACKNOWLEDGED	Alerts
`AuditEventType`	31 event types	Audit Logs, Alerts
`BackgroundJobStatus`	PENDING, RUNNING, COMPLETED, FAILED	Background Jobs
`IngestionType`	FIREFLIES, MANUAL	Meetings
`InvitationStatus`	PENDING, ACCEPTED, EXPIRED, CANCELLED	User Invitations
`MfaMethod`	TOTP, EMAIL, BACKUP_CODE, WEBAUTHN	MFA Config, MFA Challenges
`ProjectStatus`	DISCOVERY, ACTIVE, ON_HOLD, COMPLETED, ARCHIVED	Projects
`StakeholderInsightType`	QUESTION, CONCERN, STATEMENT	Stakeholder Insights
`StakeholderType`	INTERNAL, EXTERNAL, UNKNOWN	Stakeholders
`SystemVerificationType`	SYSTEM_RESET, DATABASE_RESTORE	System Verification
`TemplateType`	MEETING, BRIEF	Templates

5.6 Audit Event Types

The system tracks 31 distinct audit event types:

CATEGORY	EVENTS
Authentication	`LOGIN_SUCCESS`, `LOGIN_FAILED`, `LOGIN_LOCKED`, `LOGOUT`
Password	`PASSWORD_CHANGED`, `PASSWORD_RESET_REQUESTED`, `PASSWORD_RESET_COMPLETED`, `PASSWORD_REUSE_ATTEMPTED`
User Lifecycle	`USER_CREATED`, `USER_UPDATED`, `USER_DEACTIVATED`, `USER_REACTIVATED`, `USER_DELETED`

CATEGORY	EVENTS
Organization	`USER_ASSIGNED_TO_ORG`, `USER_ROLE_CHANGED`, `USER_INVITATION_SENT`, `USER_INVITATION_ACCEPTED`
Session	`REFRESH_TOKEN_ISSUED`, `REFRESH_TOKEN_REVOKED`, `SESSION_REVOKED_BY_LIMIT`
MFA	`MFA_ENABLED`, `MFA_DISABLED`, `MFA_VERIFIED`, `MFA_FAILED`, `MFA_BACKUP_CODE_USED`, `MFA_RECOVERY_REQUESTED`, `MFA_RECOVERY_COMPLETED`, `MFA_ADMIN_OVERRIDE`, `MFA_GRACE_PERIOD_RESET`
Device Trust	`DEVICE_TRUSTED`, `DEVICE_REVOKED`
System Health	`INSTANCE_HEALTH_THRESHOLD`, `DATABASE_HEALTH_THRESHOLD`
Application	`MEETING_DELETED`, `CUSTOMER_APPROVAL_CHANGED`

---

6. Configuration Management

6.1 Core Configuration (`app/server/src/config/core/`)

All configuration uses the NestJS `registerAs()` pattern with typed interfaces and startup validation. Environment variables are centralized in config files rather than scattered `process.env` calls:

CONFIG FILE	PURPOSE	KEY PARAMETERS
`app.config.ts`	Application settings	`PORT`, `CORS_ORIGIN`, `LOG_LEVEL`, `APP_NAME`, `NODE_ENV`, `CLIENT_URL`, `DB_MIGRATION_ON`
`api.config.ts`	API versioning	`prefix` ("api"), `defaultVersion` ("1"), URI versioning (`/api/v1/...`)
`auth.config.ts`	JWT and session settings	`JWT_SECRET`, `JWT_EXPIRES_MINUTES` (15), `JWT_REFRESH_EXPIRES_DAYS` (7), `MAX_CONCURRENT_SESSIONS` (5), `SESSION_IDLE_TIMEOUT_MINUTES` (60)
`mfa.config.ts`	MFA settings	`MFA_ISSUER`, `MFA_DEVICE_TRUST_DAYS` (14), `MFA_MANDATORY_ROLES` (Admin,SuperAdmin), `MFA_GRACE_PERIOD_DAYS` (7), `MFA_MAX_VERIFICATION_ATTEMPTS` (5), `MFA_EMAIL_CODE_EXPIRES_MINUTES` (10)
`oauth.config.ts`	OAuth providers	Google (`CLIENT_ID`, `CLIENT_SECRET`, `CALLBACK_URL`), Microsoft (`CLIENT_ID`, `CLIENT_SECRET`, `CALLBACK_URL`, `TENANT_ID`)
`ldap.config.ts`	LDAP authentication	`LDAP_URL`, `LDAP_BIND_DN`, `LDAP_BIND_CREDENTIALS`, `LDAP_SEARCH_BASE`, `LDAP_SEARCH_FILTER`

CONFIG FILE	PURPOSE	KEY PARAMETERS
`encryption.config.ts`	Encryption key management	`ENCRYPTION_KEYS_FILE` (external JSON), versioned keys (pii/mfa/hipaa), `KEY_ROTATION_BATCH_SIZE` (100), `KEY_ROTATION_DELAY_MS` (10), `KEY_ROTATION_MAX_RETRIES` (3)
`smtp.config.ts`	Email transport	`SMTP_HOST`, `SMTP_PORT` (587), `SMTP_SECURE`, `SMTP_USERNAME`, `SMTP_PASSWORD`, `SMTP_FROM_EMAIL`
`database.config.ts`	Database connection	`DATABASE_URL`, pool: `max` (20), `min` (2), `idleTimeoutMillis` (30000), `connectionTimeoutMillis` (5000), keepAlive
`redis.config.ts`	Redis connection	`REDIS_HOST`, `REDIS_PORT` (6379), `REDIS_PASSWORD`, `REDIS_TLS`, `REDIS_ENABLED`, key prefixes for throttle and token blacklist
`logging.config.ts`	Request logging	`LOG_REQUEST_BODY`, `LOG_RESPONSE_BODY`, `LOG_MAX_BODY_SIZE` (10240), `LOG_TRUNCATE_AFTER` (2000), content type exclusions
`retention.config.ts`	Data retention	`RETENTION_AUDIT_LOG_DAYS` (365), `RETENTION_AUDIT_SECURITY_DAYS` (730), `RETENTION_SOFT_DELETE_DAYS` (90), `RETENTION_BATCH_SIZE` (1000)
`siem.config.ts`	SIEM integration	`SIEM_PROVIDER` (cloudwatch/azure/gcp/datadog), batch/flush/retry settings, provider-specific configuration
`leader.config.ts`	Leader election	`LEADER_ELECTION_ENABLED`, `LEADER_RETRY_INTERVAL_MS` (10000), `LEADER_LOCK_KEY` (advisory lock ID)
`files.config.ts`	File storage	`FILES` directory path (default: `./files`)

6.2 App Configuration (`app/server/src/config/app/`)

CONFIG FILE	PURPOSE	KEY PARAMETERS
`fireflies.config.ts`	Fireflies.ai integration	`FIREFLIES_API_KEY`, `FIREFLIES_API_URL` (GraphQL endpoint)

6.3 Configuration Loading

All configs are aggregated in `app/server/src/config/index.ts` and loaded via `ConfigModule.forRoot({ isGlobal: true, load: configs })`, making typed configuration accessible throughout the application via `ConfigService.get()`.

6.4 Encryption Key File Structure

The encryption configuration requires an external JSON file (referenced by `ENCRYPTION_KEYS_FILE`) with validated structure:

- Three key types: `pii`, `mfa`, `hipaa`
- Each type contains versioned keys (`v1`, `v2`, etc.) with a `current` pointer
- Each version has a 64-character hex `key` and a unique alphanumeric `keyId`
- All key IDs must be globally unique across the entire file
- Strict validation at startup: missing keys, invalid formats, or duplicate IDs cause immediate failure

---

7. Admin Portal Architecture

7.1 Overview

Independent full-stack application (`/admin`) on separate ports (server: 4000, client: 4100). Provides infrastructure management, security scanning, database migration, and operational tooling for the main application.

7.2 Authentication and Authorization

ASPECT	IMPLEMENTATION
Identity Provider	Auth0 with RS256 JWT (validated against JWKS endpoint)
Guard Pattern	`@Roles('admin')` for mutating endpoints, `@Public()` to skip auth
WebSocket Auth	JWT passed via query parameter `?token=`, validated server-side
Auth Disable	Auth disabled when `AUTH0_DOMAIN` env var is empty (development mode)
Client SDK	`@auth0/auth0-react` with `Auth0Provider`, localStorage cache, silent token refresh
Role Extraction	Custom JWT claim `https://admin.strattdev.co/roles` (configured via Auth0 Post Login Action)

7.3 AWS Credential Exchange

ENDPOINT	PURPOSE
`GET /api/auth/aws-credentials`	Maps Auth0 roles to IAM roles via STS `AssumeRoleWithWebIdentity`

- `admin` role maps to `AWS_ADMIN_ROLE_ARN`
- All other roles map to `AWS_VIEWER_ROLE_ARN`
- CLI script (`admin/cli/credential-process.sh`) implements Auth0 Device Authorization flow for headless AWS credential acquisition

- Terraform module `devops/modules/aws/auth0_oidc/` provisions the AWS OIDC provider and IAM roles

7.4 Server Modules (`admin/server/src/modules/`)

MODULE	PURPOSE	KEY FILES
`auth/`	Auth0 JWT authentication, RBAC guards, AWS STS credential exchange	`auth.service.ts` , `auth.controller.ts` , `strategies/auth0-jwt.strategy.ts` , `guards/`
`ai/`	AI completion via Anthropic Claude	`ai.service.ts` , `ai.controller.ts`
`app-migration/`	Database schema merging, branch comparison, encryption-aware migration	`app-migration.service.ts` , `app-migration.controller.ts`
`app-security/`	Security test runner (7 categories)	`app-security.service.ts` , `test-runner.ts` , `tests/` (auth, MFA, RBAC, injection, API, encryption, infrastructure)
`backup/`	S3 backup with cron scheduling, retention, restore	`backup.service.ts` , `backup.controller.ts`
`instances/`	Docker container monitoring across master/slave via SSH	`instances.service.ts` , `instances.controller.ts`
`reset/`	Database table truncation with validation	`reset.service.ts` , `reset.controller.ts`
`scripts/`	Shell script execution with WebSocket streaming output	`scripts.service.ts` , `scripts.gateway.ts`
`security/`	OS-level security scanning (patches, firewall, SSH, file permissions)	`security.service.ts` , `security.controller.ts`
`ssh/`	Interactive SSH terminal with WebSocket, PostgreSQL tunnel (5434->5432)	`ssh.service.ts` , `ssh.gateway.ts`
`terraform/`	Terraform state inspection, infrastructure metadata	`terraform.service.ts` , `terraform.controller.ts`
`soc/`	SOC audit documentation management	Module for compliance documentation

7.5 Client Components (`admin/client/src/components/`)

FOLDER	COMPONENTS	PURPOSE
`auth/`	AdminRoute, AuthGuard, CallbackPage, LoginPage	Auth0 login guard and callback handling
`app-migration/`	AppMigrationPage, EncryptionKeysModal	Database migration UI with encryption key management
`app-security/`	AppSecurityPage, GuardScanPanel, SecurityScoreGauge, TestCategorySelector, TestProgressPanel, TestResultCard	Security testing dashboard with score visualization
`dashboard/`	DashboardPage, InstanceCard, SshModal	Main dashboard with instance overview and SSH access
`deployment/`	DeployPage, DeployProvisioningForm, BackupForm, BackupHistoryModal, DockerPage, DockerImagesPanel, BuildImagesForm, PushToEcrForm, UpdateImagesForm, EncryptionForm, OAuthForm, SmtpForm, ResetForm	Infrastructure deployment and configuration management
`infrastructure/`	InfrastructurePage	Infrastructure overview
`layout/`	AdminLayout, Sidebar	Application shell and navigation
`security/`	SecurityPage, SecurityScanPanel, SecurityReportPanel, SecurityReportView, ApplyFixPanel	OS security scanning and remediation
`shared/`	TerminalOutput, ConfirmationModal, ConfirmCodeModal, AppHealthBanner, InfrastructureBanner, InstanceStatusBanner, InstanceSelector	Reusable UI components
`ssh/`	AddKeyPanel	SSH key management
`soc/`	SocPage, MarkdownEditorModal	SOC documentation management

7.6 Real-Time Communication

GATEWAY	WEBSOCKET PATH	PURPOSE
`scripts.gateway.ts`	`/ws`	Shell script execution with streaming stdout/stderr
`ssh.gateway.ts`	`/ws/ssh`	Interactive SSH terminal (xterm.js client)

Both gateways validate JWT tokens passed via WebSocket query parameters.

---

8. Cross-Cutting Concerns

8.1 Event-Driven Architecture

NestJS `@nestjs/event-emitter` is used for decoupled cross-module communication:

EVENT CLASS	EMITTER	LISTENERS	PAYLOAD
`AuditLogEvent`	Auth, MFA, RBAC, Invitation, Meetings, System	AuditListener, AlertingListener	`eventType`, `userId`, `organizationId`, `targetEmail`, `ipAddress`, `userAgent`, `metadata`
`UserCreatedEvent`	AuthService	RbacListener (assigns initial role)	`userId`, `email`, `source` (registration/oauth/ldap)
`MfaVerifiedEvent`	MfaService	Auth flow completion	`userId`, `context` (IP, user agent)
`BackgroundJobCreatedEvent`	BackgroundJobsService	Job processor	`jobId`, `jobType`, `entityId`, `entityType`

8.2 Field-Level Encryption

Transparent encryption/decryption via Prisma `$extends()` middleware:

- Three key types: `pii` (personal data), `mfa` (MFA secrets), `hipaa` (health data)
- AES-256-GCM encryption with per-key-type versioning
- Deterministic mode for searchable fields (email lookups)
- Non-deterministic mode for other PII (names, IPs, user agents)
- Key rotation support with batch processing, pause/resume, and per-record error tracking
- Rotation audit trail via `key_rotation_events` and `encryption_key_audits` tables

8.3 Structured Logging

- Winston logger with configurable log levels
- Global HTTP interceptor logs all requests with sanitized bodies
- Request body logging configurable (default off in production)
- Body size limits (10KB max, 2000 char truncation)
- Content type exclusions (multipart, binary, media)
- SIEM integration with multiple providers:
    - AWS CloudWatch (active)
    - Azure Monitor (future)
    - Google Cloud Logging (future)

    - Datadog (future)
- SIEM batching (100 events), configurable flush interval (5s), retry (3 attempts)
- Sensitive field redaction in logs
- `@SkipBodyLogging()` decorator for endpoints handling credentials

8.4 Rate Limiting

NestJS `@nestjs/throttler` with global default (50 requests/minute per IP) and per-endpoint overrides:

ENDPOINT	LIMIT	WINDOW	NOTES
Global default	50	1 min	Redis-backed when available, in-memory fallback
Login	5	1 hour	Plus account lockout after configurable failures
Registration	10	1 hour	Per IP
Password reset request	3	1 hour	Plus separate per-email rate limiting
Password reset confirm	5	15 min	
Password change	5	1 hour	
Token refresh	30	1 min	
LDAP login	5	15 min	

Development mode uses elevated limits (1000) to avoid blocking during testing.

8.5 Data Retention

Automated retention policies enforced via leader-elected cron jobs:

DATA TYPE	RETENTION PERIOD	MECHANISM
Standard audit logs	365 days	Hard delete after period
Security audit logs	730 days	Extended retention for security events
Soft-deleted records	90 days	Permanent removal after grace period
Batch size	1000 records	Per cleanup cycle

8.6 Leader Election

PostgreSQL advisory lock-based leader election for distributed deployments:

- Ensures scheduled jobs (retention cleanup, health checks) run on exactly one instance
- Configurable retry interval (default 10s)

- Lock key configurable to avoid collisions across environments

8.7 HTTP Security Headers

Helmet middleware applies the following Content-Security-Policy:

DIRECTIVE	VALUE	RATIONALE
`default-src`	`'none'`	Deny all by default
`script-src`	`'self'`	Only same-origin scripts
`style-src`	`'self'`, `'unsafe-inline'`	Required for inline table styles
`font-src`	`'self'`, `data:`	Self-hosted fonts plus data URIs
`img-src`	`'self'`, `data:`, `blob:`	Self-hosted images plus data/blob URIs
`connect-src`	`'self'`	API calls restricted to same origin
`frame-ancestors`	`'self'`	Prevent clickjacking
`base-uri`	`'self'`	Prevent base tag injection
`form-action`	`'self'`	Restrict form submissions
`object-src`	`'none'`	Block plugins

8.8 CORS Configuration

CORS origin is configurable via `CORS_ORIGIN` environment variable with `credentials: true` to support httpOnly cookie transmission.

---

9. Compliance Considerations

AREA	SEVERITY	FINDING	REMEDIATION
Audit Trail	Critical	Event-driven audit logging captures 31 distinct security event types with structured metadata, IP addresses, user agents, and organization context. Audit logs are indexed for efficient compliance queries.	Maintain current implementation. Periodically review event type coverage as new features are added.
Data Encryption at Rest	Critical	Field-level AES-256-GCM encryption via Prisma middleware with three key types (pii, mfa, hipaa). Supports both deterministic (searchable) and non-deterministic modes. Versioned keys with rotation support.	Ensure encryption keys are stored securely outside the application repository. Validate key rotation

AREA	SEVERITY	FINDING	REMEDIATION
			procedures are tested regularly.
Encryption Key Management	Critical	External key file with strict startup validation (format, uniqueness, hex pattern). Version-based rotation with batch processing, pause/resume, per-record error tracking, and rotation audit trail.	Implement automated key rotation schedules. Ensure key file access is restricted at the OS level. Back up key files in a separate secure location.
Access Control (RBAC)	Critical	Casbin RBAC with 5-level priority hierarchy (SuperAdmin > Admin > Consultant > Customer > Unapproved). Organization-scoped data with JWT-embedded role context. Guard enforcement on all protected endpoints.	Review Casbin policies periodically. Ensure new modules include RBAC guards. Validate that role assignments follow least-privilege principles.
Authentication	Critical	JWT with httpOnly secure cookies, OAuth (Google/Microsoft), LDAP integration, account lockout after failed attempts, password history enforcement, session management with concurrent session limits.	Ensure JWT_SECRET is rotated periodically. Monitor failed login patterns. Verify OAuth callback URLs are restricted to known domains.
JWT Secret Validation	Critical	Server refuses to start without JWT_SECRET defined, preventing insecure operation.	Maintain mandatory startup check. Ensure secret is generated with sufficient entropy (256+ bits).
Multi-Factor Authentication	High	TOTP, email OTP, backup codes, trusted devices with expiration, mandatory MFA for Admin and SuperAdmin roles with configurable grace period (default 7 days). MFA secrets encrypted with dedicated mfa key type.	Enforce mandatory MFA for all privileged roles. Monitor MFA grace period usage. Review trusted device expiration policy (default 14 days).
Session Management	High	JWT token blacklist (database + Redis), concurrent session limits (default 5), idle timeout (60 min), refresh token tracking with	Monitor for unusual session patterns. Consider reducing concurrent session

AREA	SEVERITY	FINDING	REMEDIATION
		device info, session revocation endpoints. Token invalidation timestamp per user.	limit for high-privilege roles.
Rate Limiting	High	Per-endpoint throttling with configurable limits. Login-specific lockout. Separate password reset rate limiting per-email. Redis-backed when available with in-memory fallback. Global default: 50 req/min per IP.	Review rate limits periodically against attack patterns. Ensure Redis is enabled in production for distributed rate limiting.
GDPR Compliance	High	User deletion endpoint (right-to-erasure) via SuperAdmin role with cascade delete on related records. Soft delete with 90-day retention before permanent removal. Audit log user references set to null on delete.	Document data subject access request (DSAR) procedures. Validate cascade delete covers all user-linked data.
Data Retention	High	Configurable retention periods: audit logs (365 days), security audit logs (730 days), soft-deleted records (90 days). Leader-elected cron jobs ensure single-instance execution. Batch processing (1000 records per cycle).	Review retention periods against regulatory requirements. Ensure retention jobs are monitored for failures.
Input Validation	High	Global ValidationPipe with `whitelist: true` and `forbidNonWhitelisted: true`. DTO-based validation with class-validator decorators on all endpoints. Parameterized queries via Prisma ORM (SQL injection prevention).	Ensure all new endpoints use DTO validation. Review custom query construction for injection risks.
Structured Logging and SIEM	High	Winston logger with SIEM integration (CloudWatch active; Azure, GCP, Datadog planned). Request/response body logging with size limits and sensitive field redaction. Content type exclusions for binary data.	Enable SIEM integration in production. Review redaction rules for completeness. Implement log alerting for security events.
Password Policy	High	Configurable policies including minimum length, complexity requirements, and history. Password reuse prevention with `password_history` table. Reset rate limiting per-email.	Enforce minimum 12-character passwords with complexity. Set password history to at least 12 previous passwords.

AREA	SEVERITY	FINDING	REMEDIATION
Backup and Recovery	High	Main app: database backup/restore with email verification codes (SuperAdmin only). Admin portal: S3 backup with cron scheduling, retention policies, and restore capability.	Test restore procedures regularly. Encrypt backups at rest. Validate backup integrity with checksums.
Admin Portal Security	High	Auth0 JWT (RS256/JWKS validation), role-based guards, AWS STS credential exchange mapping roles to IAM permissions. WebSocket JWT validation. Auth can be disabled via empty env var for development.	Ensure AUTH0_DOMAIN is always set in production. Restrict Auth0 tenant access. Review IAM role policies for least-privilege.
Multi-Tenancy Isolation	High	Organization-scoped data with JWT-embedded org context. Casbin domain-based policies. User-organization-role junction table enforces membership. App data requires organization foreign key.	Audit cross-tenant data access patterns. Add automated tests for tenant isolation. Verify all app-level queries include organization filter.
Error Handling	Medium	Consistent AppException format with machine-readable codes. Global exception filter prevents stack trace leakage in production. Unhandled exceptions logged server-side with full context.	Review exception filter for information disclosure in edge cases. Ensure all custom exceptions use the AppException base class.
Separation of Concerns	Medium	Core vs app module boundary. Controller-service-repository pattern. No barrel files. No `forwardRef()`. Only PrismaModule and LoggerModule are global. Event emitter for cross-module communication.	Enforce module boundaries in code review. Monitor for direct service-to-service imports that should use events.
API Security	Medium	URI versioning (`/api/v1/...`), auto-generated OpenAPI documentation, CORS with configurable origin, SameSite cookie protection, Helmet security headers with strict CSP. Swagger UI disabled in production.	Validate CORS_ORIGIN is restricted to known domains in production. Review CSP `unsafe-inline` usage for style-src.

AREA	SEVERITY	FINDING	REMEDIATION
Infrastructure Monitoring	Medium	Instance health tracking (CPU/RAM/disk per instance), database health monitoring, leader-elected health checks, threshold-based alerting via audit event system.	Set appropriate threshold values for alerts. Ensure monitoring covers all deployed instances. Integrate with external monitoring (e.g., CloudWatch alarms).
Security Testing	Medium	Admin portal includes automated security test runner covering 7 categories: authentication, MFA, RBAC, injection, API, encryption, and infrastructure. Guard scan analysis tooling.	Run security tests as part of CI/CD pipeline. Expand test coverage as new security controls are added.
Content Security Policy	Medium	Helmet CSP restricts script sources to `'self'` and blocks object/plugin content. `frame-ancestors: 'self'` prevents clickjacking. `upgrade-insecure-requests` enforces HTTPS.	Remove `'unsafe-inline'` from `style-src` when feasible. Consider adding nonce-based CSP for stricter script control.
WebSocket Security	Medium	Admin SSH and script execution gateways validate JWT via query parameter. Connection established only after token verification.	Consider moving JWT from query parameter to WebSocket protocol headers to prevent token leakage in server logs.
Background Job Processing	Low	Asynchronous AI processing with status tracking (PENDING/RUNNING/COMPLETED/FAILED), event-driven execution, error capture with summaries.	Monitor failed job rates. Implement dead-letter handling for persistently failing jobs.
Configuration Management	Low	All 16 config files use NestJS `registerAs()` pattern with typed interfaces. Config loaded globally at startup. No scattered `process.env` calls in business logic.	Maintain centralized config pattern. Audit environment variables for unused or deprecated entries.