Technical Specifications

Corporate Strategy & Intelligence Dossier

Prepared for	Project-X Executive Leadership Team
Prepared by	Strategy & Compliance Office
Document date	October 24, 2025
Classification	Strictly Confidential – Do Not Distribute

CONFIDENTIAL

This document contains proprietary, privileged, and confidential information belonging to Project X Holdings. It is provided solely for the designated recipients and must not be copied, distributed, or disclosed to any third party without prior written consent. By accepting this document you agree to maintain its confidentiality and to use the information only for the purpose for which it was provided.

© 2025 Technical Specifications - Innovista. All rights reserved.

Table of Contents

CONFIDENTIAL

Confidential – Attorney-Client Privileged & Proprietary
© 2025 Technical Specifications - Innovista. All rights reserved.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

1. System Architecture

TL;DR

This section provides a complete overview of the platform's technical architecture — describing how the frontend, backend, and infrastructure layers interact within a secure, modular, and scalable design.

It details the **JavaScript-only technology stack (no TypeScript used anywhere)**, explains component-level responsibilities, outlines data flow between layers, and explicitly notes that both **PostgreSQL (database)** and **MinIO (object storage)** are **externally hosted**.

The objective is to provide developers and architects with a clear, end-to-end understanding of how the system is structured, deployed, and maintained.

CONFIDENTIAL

1.1 Purpose and Scope

This document defines the **technical architecture** of the AI Governance Platform, focusing on how all components — frontend, backend, and infrastructure — work together to deliver a secure, efficient, and scalable governance system.

It serves as a foundation for all engineering teams, outlining system structure, technology choices, integration points, and deployment principles.

Note: The entire platform is built using **JavaScript only**, and **TypeScript will not be used anywhere** in the codebase.

This applies to both frontend and backend layers to maintain simplicity, faster development cycles, and unified syntax across all components.

1.2 Tech Stack Overview

© 2025 Technical Specifications - Innovista. All rights reserved.

Technology	*Purpose*
Postgres	Primary relational database **hosted externally** (connected via .env connection string).
Express.js	Server framework for REST APIs written entirely in JavaScript.
React.js (JavaScript)	Client framework for user interface — built exclusively in JavaScript with no TypeScript.
Vite	Lightweight build tool and dev server for fast frontend development (used instead of Next.js).
React Router DOM	Client-side routing for single-page application (SPA) navigation.
Axios	HTTP client for communication between frontend and backend.
Node.js	JavaScript runtime environment for server-side execution.
TailwindCSS + shadcn	Utility-first CSS framework and component library for consistent, accessible UI design.
Lucide React Icons	Modern icon set integrated with shadcn/ui components.
Tiptap Editor	Headless WYSIWYG editor integrated with React. Supports direct uploads to **MinIO** for all media (videos, images, and documents).
Prisma ORM	Database schema and management tool for Postgres, written in JavaScript.
JWT	Authentication strategy using JSON Web Tokens for securing API routes and sessions.
Casbin	Policy-based role and permission control integrated into Express.js.
dotenv	Environment variable management for configuration across backend and frontend.
CORS Middleware	Enables secure cross-origin communication between the Vite frontend and Express backend.
Nodemailer	Server-side email delivery and notification service.
Winston + Morgan	Structured JSON logging for server monitoring and debugging.
Swagger (OpenAPI)	API documentation for backend routes.
MinIO	**Externally hosted object storage** for all file uploads, using presigned URLs for secure access.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

1.3 Code and Project Structure

This section defines how the project repository and its codebase are organized across frontend, backend, shared components, and infrastructure.

The goal is to maintain a consistent, modular, and scalable folder structure that aligns with the JavaScript-only architecture.

Root Repository Structure

The project repository is organized at the root level as follows:

- **client/** – Contains the React.js frontend application (built with Vite).
- **server/** – Contains the Express.js backend API service.
- **shared/** – Common logic, constants, and utilities shared between client and server.
- **infra/** – Infrastructure-as-Code, deployment automation, and operational configuration for DevOps and cloud environments.
- **docs/** – Product requirement documents (PRDs), feature specifications, and user stories.
- **.env.example** – Template file for environment variable configuration.

Organizational Guidelines

- The repository follows a **feature-based organization** to improve modularity and developer ownership.
- All components related to a specific business feature are self-contained within their respective domain.
- Shared integrations (Nodemailer, MinIO, Casbin) are located under **server/src/integrations/** for reusability.
- All DevOps and infrastructure configurations reside in the **infra/** directory to centralize provisioning, automation, and environment management.

Directory Structure

Frontend (React Client)

- Source code located in client/src/features/<feature>
- Each feature folder includes its own components, hooks, styles, and Axios API handlers.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Backend (Express Server)

- Source code organized in server/src/modules/<feature>
- Each module includes its own controller, service, route, and validation logic.
- Shared integrations live in server/src/integrations and include:
- **Nodemailer** for email delivery
- **MinIO** for file storage
- **Casbin** for policy-based access control

Infrastructure (infra Directory)

- **infra/terraform/** – Terraform configurations and reusable modules.
- **infra/kubernetes/** – Kubernetes manifests or Helm charts for deployment.
- **infra/scripts/** – Automation and deployment scripts.
- **infra/monitoring/** – Prometheus, Grafana, and alert configurations.
- **infra/policies/** – Policy-as-Code definitions for CI/CD compliance checks.
- **infra/backups/** – Backup management for PostgreSQL and MinIO.

Shared Logic

Common libraries or utilities used by both frontend and backend reside in the shared/ directory.

This includes constants, validation schemas, and helper functions to avoid duplication.

Environment Configuration

- A single .env file is used for all services (frontend, backend, and shared).
- Environment variables control connections to:
- Externally hosted **PostgreSQL database**
- Externally hosted **MinIO storage**
- Email and authentication services
- The .env.example file serves as the template for new environments.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

This structure standardizes development, simplifies CI/CD automation, and ensures consistent code organization across all modules.

1.4 Architectural Overview

The system follows a **service-oriented JavaScript architecture** composed of independent, containerized modules.

All services communicate over RESTful APIs secured by HTTPS.

Key Characteristics:

- 100% **JavaScript-based** — no TypeScript or language transpilation used.
- **PostgreSQL** and **MinIO** are **externally hosted** and connected via secure credentials.
- Modular, containerized components for maintainability and scaling.
- Unified environment management through .env configuration files.

1.5 Core System Components

Frontend Application

- Developed with **React.js (JavaScript)** using Vite as the build tool.
- Implements a single-page architecture with client-side routing.
- Provides dashboards, reports, and compliance workflows.
- Communicates with backend APIs through Axios.
- Uses TailwindCSS and shadcn for UI consistency and accessibility.
- Integrates **Tiptap** for editable content with media uploads to **external MinIO storage**.
- Maintains authentication state with JWT.

API Server (Backend)

- Built using **Express.js (JavaScript)** and Node.js.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Hosts RESTful APIs for authentication, compliance scoring, framework mapping, and audit tracking.
- Utilizes Casbin for access control and JWT for authentication.
- Includes CORS middleware, structured error handling, and validation logic.
- Serves OpenAPI/Swagger documentation for all routes.

Database Layer (Externally Hosted)

- **PostgreSQL is hosted externally** on a managed cloud service.
- Managed via Prisma ORM for schema, migrations, and queries.
- Stores users, roles, frameworks, compliance results, and evidence metadata.
- Secure SSL connection configured via environment variables.
- Backup and access control managed at the hosting provider level.

Object Storage (Externally Hosted MinIO)

- **MinIO is hosted externally** as a secure object storage service.
- Used for document, image, and video storage.
- Files uploaded via presigned URLs, linked to metadata in PostgreSQL.
- Encryption at rest and in transit enabled by default.

Logging and Monitoring

- **Winston** and **Morgan** provide structured logs for backend activity.
- Logs integrated with application lifecycle and API monitoring tools.
- Optional integration with external observability systems.

1.6 Data Flow

1. **User Authentication** – Frontend authenticates via JWT through Express APIs.

2. **Frontend Interaction** – React SPA manages routes with React Router DOM and communicates via Axios.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

3. **Backend Processing** – Express handles API requests, Casbin enforces roles, Prisma executes queries.

4. **Database Transactions** – Data persisted in **externally hosted PostgreSQL** with referential integrity.

5. **File Uploads** – Files uploaded directly to **externally hosted MinIO** using presigned URLs.

6. **Notifications** – Nodemailer sends email alerts for user and compliance events.

1.7 Deployment Architecture

Environment Layers

- **Development:** Local Docker setup connected to external Postgres and MinIO services.
- **Staging:** Pre-production testing environment mirroring production.
- **Production:** Fully containerized cloud deployment connected securely to **external Postgres** and **MinIO endpoints**.

Containerization

- Each major service (client, server) runs in separate containers.
- Managed via Docker Compose or Kubernetes.
- Stateless architecture enables horizontal scaling.

Configuration and Security

- Environment variables managed through .env files.
- Sensitive keys stored securely in cloud secret management tools.
- HTTPS enforced across all external communications.
- JWT and Casbin policies consistently applied for access control.

Networking

- All API traffic flows through secure HTTPS endpoints.
- Database and MinIO connections restricted to authorized IP ranges.
- CORS configured to allow only approved frontend origins.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

1.8 Cross-Cutting Concerns

- **Language Standardization:** 100% JavaScript — no TypeScript used anywhere in the platform.
- **Security:** JWT authentication, Casbin-based RBAC, and SSL-secured connections.
- **Performance:** Optimized builds with Vite, efficient queries via Prisma ORM.
- **Reliability:** External Postgres and MinIO ensure durability, redundancy, and backup coverage.
- **Scalability:** Stateless backend, containerized components, and distributed deployment model.
- **Monitoring:** Structured JSON logs with Winston and Morgan for traceability and audit readiness.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

2. Backend Architecture & APIs

TL;DR

This section defines the backend architecture of the AI Governance Platform, explaining how the Node.js and Express.js backend is structured, organized, and connected to the external database and storage services.

It covers the folder structure, core backend microservices (Auth, Governance Engine, Frameworks, Evidence Repository, Notifications, and Tasks), database schema overview, API design standards, error handling, and documentation practices.

The objective is to maintain consistency, security, and scalability across all backend services in a **JavaScript-only implementation** (no TypeScript).

CONFIDENTIAL

2.1 Purpose and Overview

The backend architecture provides the logic, integrations, and data management foundation for the platform.

It processes all API requests, executes compliance logic, manages authentication and evidence, and communicates with external services such as MinIO and Nodemailer.

The backend is entirely written in **JavaScript (Node.js + Express.js)** and interacts with an **externally hosted PostgreSQL** database.

It is modular, containerized, and scalable, designed for independent feature development and simplified maintenance.

2.2 Folder Structure and Conventions

© 2025 Technical Specifications - Innovista. All rights reserved.

The backend follows a **feature-based modular structure**, ensuring clear separation of concerns and maintainability.

Root-level directories

- **server/src/modules/** – Feature-specific backend modules.
- **server/src/integrations/** – Shared integrations (MinIO, Nodemailer, Casbin).
- **server/src/middleware/** – Common middleware for authentication, validation, and logging.
- **server/src/config/** – Application configuration and connection management.
- **server/src/utils/** – Helper functions, constants, and reusable logic.
- **server/src/routes/** – API route definitions and module registration.
- **server/src/app.js** – Express app initialization.
- **server/src/server.js** – Backend server entry point.

Naming Conventions

- Filenames use lowercase with hyphens (e.g., user-controller.js, task-service.js).
- Environment variables use uppercase snake case (e.g., DB_URL, MINIO_ACCESS_KEY).
- Controllers, services, and routes follow consistent naming to simplify discovery and testing.

2.3 Core Backend Services

Auth Service

Purpose: Handle authentication, authorization, and user session management.

Responsibilities:

- Manage JWT authentication tokens and session lifecycle.
- Enforce role-based access control using Casbin.
- Handle registration, login, and password recovery flows.
- Use Nodemailer for password reset and verification emails.

Key Endpoints:

- /auth/login

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- /auth/register
- /auth/logout
- /auth/refresh

Governance Engine

Purpose: Core engine that processes compliance data, evaluates checks, and computes control-level metrics.

Responsibilities:

- Execute defined checks based on governance rules.
- Aggregate results into measurable controls.
- Calculate compliance scores and risk levels.
- Trigger remediation workflows for failed checks.
- Log and store audit events in PostgreSQL.

Key Functions:

- Check validation and aggregation.
- Control-level scoring.
- Risk categorization and report generation.

Framework Service

Purpose: Manage governance frameworks, control structures, and external compliance mappings.

Responsibilities:

- CRUD operations for frameworks and controls.
- Map internal controls to external standards (EU AI Act, ISO 42001, NIST AI RMF).
- Maintain framework versions and update tracking.
- Provide metadata for reporting and dashboard modules.

Key Endpoints:

- /frameworks

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- /frameworks/:id/controls
- /frameworks/:id/mappings

Evidence Repository

Purpose: Centralized evidence management system for file uploads and compliance documentation.

Responsibilities:

- Interface with **externally hosted MinIO** for storage.
- Generate presigned URLs for secure uploads and downloads.
- Maintain metadata linkage between files and controls.
- Track file versioning and audit trails.
- Encrypt and validate file access permissions.

Key Endpoints:

- /evidence/upload
- /evidence/:id/download
- /evidence/:id/metadata

Notification Service

Purpose: Manage platform notifications and communication events.

Responsibilities:

- Send automated and triggered email notifications using Nodemailer.
- Support system alerts for new tasks, completed remediations, and audit updates.
- Manage notification templates and scheduling.
- Store notification delivery logs in PostgreSQL.

Key Endpoints:

- /notifications/send
- /notifications/templates
- /notifications/logs

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Task Service

Purpose: Manage remediation tasks, user assignments, and progress tracking.

Responsibilities:

- Create, assign, and track compliance tasks generated by failed checks.
- Maintain task status (open, in progress, resolved).
- Enable escalation and reassignment for overdue tasks.
- Link tasks with evidence, controls, and framework items.
- Provide endpoints for reporting and analytics.

Key Endpoints:

- /tasks
- /tasks/:id
- /tasks/:id/status
- /tasks/:id/reassign

2.4 Database Schema Overview

The backend interacts with an **externally hosted PostgreSQL** database managed through Prisma ORM.

Schema migrations are automated as part of deployment. The managed provider exposes a dedicated Project X schema, and our CI/CD pipeline submits Prisma migration bundles to the host's change queue so that RBAC tables (e.g., auth_policies) and other structures remain under our governance despite the external hosting model.

Core Tables

- **users** – User profiles and credentials.
- **roles** – Access control definitions for Casbin policies.
- **frameworks** – Governance framework definitions.
- **controls** – Individual governance control records.
- **checks** – Validation results linked to controls.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **evidence** – Metadata for documents stored in MinIO.
- **notifications** – Outgoing communication records.
- **tasks** – Remediation and workflow assignments.
- **audit_logs** – Immutable log of compliance activities.

Schema Design Principles

- UUIDs as primary keys.
- Foreign key integrity between related entities.
- Indexes on high-query columns (e.g., `user_id`, `framework_id`).
- Automatic timestamps for tracking creation and updates.
- Soft deletes via status flags where appropriate.

2.5 API Specification Standards

The backend follows **REST-based APIs** documented through **OpenAPI 3.0**.

All endpoints use secure HTTPS communication and JWT-based authorization.

Design Principles

- Resource-oriented endpoints with predictable patterns.
- Consistent HTTP methods: GET, POST, PUT/PATCH, DELETE.
- Unified JSON response structure containing:
- `status`
- `message`
- `data`
- `error` (if applicable)

Security

- JWT validation on every protected route.
- Casbin middleware enforces role and permission policies.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- HTTPS enforced for all communications.
- CORS configured for approved frontend origins only.

Documentation

- All endpoints are defined and auto-documented via Swagger (OpenAPI).
- Live documentation served under /api-docs.
- Updated automatically with new releases.

2.6 Error Handling and Logging

The backend uses unified error management and structured logging for observability.

Error Handling

- Global middleware captures and normalizes all errors.
- Validation errors return HTTP 400 with field-level messages.
- Unauthorized and forbidden requests return 401 and 403 respectively.
- Unhandled exceptions return standardized 500 responses.

Logging

- **Winston** and **Morgan** manage structured JSON logs.
- Logs capture route, status, execution time, and user context.
- Sensitive data never written to logs.
- Logs aggregated for external monitoring and auditing.

2.7 Versioning and Documentation

API Versioning

- All endpoints versioned under /api/v1/, /api/v2/ for compatibility.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Deprecated APIs remain accessible for legacy clients until migration.

Documentation and Change Management

- API documentation generated automatically with Swagger.
- Each release includes updated OpenAPI definitions and changelogs.
- Internal wiki tracks module design decisions and dependency updates.

CONFIDENTIAL

Confidential – Attorney-Client Privileged & Proprietary
© 2025 Technical Specifications - Innovista. All rights reserved.

3. Frontend Architecture

TL;DR

This section defines the frontend architecture of the AI Governance Platform.

It describes how the **React.js (JavaScript)** application is structured, how it interacts with backend APIs, and how the user interface is organized into modular, maintainable components.

The frontend follows a **feature-based architecture**, built entirely in **JavaScript with no TypeScript**, using **Vite**, **React Router DOM**, **TailwindCSS**, and **shadcn/ui**.

The objective is to ensure a consistent, secure, and performant client-side application that integrates seamlessly with the backend services.

CONFIDENTIAL

3.1 Purpose and Overview

The frontend provides the primary interaction layer between users and the platform's governance ecosystem.

It is built using **React.js** with **Vite** for a fast development experience and optimized builds.

The entire client application is developed in **JavaScript only** — no TypeScript — ensuring consistent syntax and a simpler build pipeline across the engineering team.

The frontend consumes REST APIs exposed by the Express.js backend and provides dashboards, compliance workflows, scorecards, reports, and administrative features.

Its design focuses on **usability, performance, modularity, and accessibility**.

3.2 Project and Folder Structure

© 2025 Technical Specifications - Innovista. All rights reserved.

The frontend codebase follows a **feature-based organization** aligned with backend modules for maintainability.

Root-level directories

- **client/src/features/** – Feature-specific folders (e.g., auth, dashboard, evidence, framework, tasks).
- **client/src/components/** – Reusable shared components (buttons, modals, inputs, charts).
- **client/src/layouts/** – Application layouts (main layout, dashboard layout, auth layout).
- **client/src/hooks/** – Custom React hooks for data fetching, state handling, and API integration.
- **client/src/context/** – Global context providers (auth, theme, notifications).
- **client/src/routes/** – Route definitions managed via React Router DOM.
- **client/src/utils/** – Utility functions, constants, and configuration helpers.
- **client/src/styles/** – Tailwind configuration, global styles, and theme definitions.
- **client/src/assets/** – Icons, logos, and images.
- **client/src/app.js** – Main React application initializer.
- **client/src/main.jsx** – Vite entry point and rendering logic.

Naming Conventions

- All files use lowercase with hyphens (dashboard-page.jsx, auth-context.js).
- Component names use PascalCase (DashboardCard, UserMenu).
- Shared utilities and hooks use camelCase.

3.3 UI Architecture and Component Design

The frontend follows a **modular, component-driven design** using the principles of **Atomic Design** and feature encapsulation.

Component Hierarchy

1. **Atoms:** Smallest UI elements (buttons, inputs, labels).
2. **Molecules:** Combined components that serve specific functions (search bars, dropdowns).
3. **Organisms:** Complex interface components (navbars, side panels, forms).

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

4. **Templates:** Page-level structures defining UI layout.

5. **Pages:** Final compositions linked to routes.

Design Philosophy

- Reusable and independent components for scalability.
- Shared components reside under client/src/components/ for reuse.
- Each feature folder manages its own page components, local states, and styles.
- Components built with **shadcn/ui** and **TailwindCSS** ensure consistent styling and accessibility.

3.4 State Management

State management is handled through a combination of **React Context API** and **custom hooks** for simplicity and performance.

Redux or Zustand may be introduced later if large-scale global state synchronization becomes necessary.

Current State Layers

- **Auth Context:** Manages JWT tokens, user sessions, and role data.
- **Theme Context:** Controls light/dark mode and theme persistence.
- **Notification Context:** Handles in-app notifications and alerts.
- **Feature-Level Local State:** Managed using React useState and useReducer within feature components.

This hybrid approach balances simplicity with scalability while maintaining JavaScript-only consistency.

3.5 Theming and Design System

The UI follows a unified design language based on **TailwindCSS** and **shadcn/ui**.

Design System Components

- **Typography:** Tailwind-based scalable font system.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **Colors:** Semantic colors mapped to compliance domains (green = compliant, red = risk, amber = pending).
- **Components:** Prebuilt shadcn components (cards, buttons, tables, alerts, modals).
- **Icons: Lucide React Icons** for consistent and modern visual cues.

Theme Management

- Theme configuration stored in context for runtime switching.
- Follows accessibility standards (WCAG AA).
- Supports dark and light themes, customizable per user preference.

3.6 API Integration and Data Flow

The frontend interacts with the backend via REST APIs using **Axios** as the HTTP client.

All API calls are abstracted into dedicated service files for maintainability.

Data Flow

1. User performs an action (e.g., submitting evidence, viewing compliance reports).
2. Axios sends a request to the backend’s Express API.
3. JWT token attached in headers for authorization.
4. Response handled and normalized through a shared interceptor.
5. Data passed to the relevant feature component or global context.
6. UI updates reflect new data through re-rendering.

API Layer

- API endpoints defined in client/src/features/<feature>/api.js.
- Interceptors handle error responses and token expiration.
- Common configuration stored in client/src/utils/api-config.js.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

3.7 Security and Accessibility

The frontend enforces multiple layers of security to protect user data and prevent misuse.

Security Measures

- **JWT-based authentication** for all protected routes.
- **CORS** policies configured on the backend for authorized origins only.
- **CSRF protection** via secure token headers.
- **Content Security Policy (CSP)** enforced in deployment environments.
- **Input validation** on both client and server layers.
- **Session timeout** and token refresh policies to prevent stale sessions.

Accessibility (A11y)

- All components follow accessibility guidelines.
- Semantic HTML elements for screen reader compatibility.
- Keyboard navigation support across interactive elements.
- High-contrast themes and focus states for visibility.

3.8 Localization and Configuration

Localization ensures that the interface can adapt to different languages and regional contexts as the platform scales.

Localization Model

- Text content managed via JSON language files (e.g., `en.json`, `fr.json`).
- Locale switching supported via user preferences stored in the database.
- Dates, currencies, and numeric formats localized using standard JS libraries.

Configuration

- Environment variables managed via `.env` for frontend.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Configuration includes API base URL, analytics keys, and theme defaults.
- No secrets stored client-side; sensitive keys managed server-side.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

4. Database Design

TL;DR

This section defines the database architecture, schema design, and operational principles for the AI Governance Platform.

It explains how the **externally hosted PostgreSQL** database is structured, managed, and secured using **Prisma ORM**.

The database is fully relational, optimized for governance data models including users, frameworks, controls, evidence, and audit logs.

The objective is to ensure integrity, performance, and scalability while maintaining alignment with compliance and data protection standards.

CONFIDENTIAL

4.1 Purpose and Overview

The database design defines the persistent data model for the AI Governance Platform.

It stores and manages all core entities — user accounts, governance frameworks, compliance checks, evidence, and audit data.

The system uses **PostgreSQL** hosted externally, providing reliability, scalability, and compliance readiness.

All data access is abstracted through **Prisma ORM**, ensuring schema consistency and version-controlled migrations. Even though the PostgreSQL cluster is externally hosted, Project X operates within a dedicated schema and executes Prisma migrations through the provider's managed change pipeline, giving the team full control over table structure and indices.

The database adheres to relational design principles with normalized data models and clear referential relationships between entities.

4.2 Database Architecture

© 2025 Technical Specifications - Innovista. All rights reserved.

The database follows a **modular, schema-driven structure** optimized for compliance workloads.

It supports high read and moderate write operations, aligning with the platform's continuous governance monitoring model.

Key Architectural Features

- **Externally hosted PostgreSQL instance** for managed security, backups, and replication.
- **Dedicated Project X schema** provisioned on the managed instance, with migrations applied via Prisma through an automated change queue so tables such as auth_policies remain under first-party control.
- **Prisma ORM layer** for data modeling, migrations, and query abstraction.
- **Normalized schema (3NF)** to avoid data duplication and ensure consistency.
- **Role-based access** to restrict data operations by service type.
- **Connection pooling** managed via Prisma for efficient query handling.
- **Time-based partitioning** for large tables such as audit logs and evidence metadata.

4.3 Schema Structure and Core Tables

The platform's schema is composed of interconnected entities that represent governance, compliance, and operational data.

Users and Roles

- **users** – Stores account data, authentication details, and basic profile info.
- **roles** – Defines user access levels (admin, compliance officer, engineer, auditor).
- **permissions** – Maps actions and scopes used by Casbin for policy enforcement.
- **relationships:** One-to-many between roles and users.

Frameworks and Controls

- **frameworks** – Contains framework metadata (title, version, region, source).
- **controls** – Stores governance control definitions linked to frameworks.
- **mappings** – Associates controls with multiple frameworks for interoperability.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **relationships:** One-to-many between frameworks and controls; many-to-many between frameworks through mappings.

Probes and Checks

- **probes** – Defines data collectors or integrations fetching evidence.
- **checks** – Represents specific validation rules executed by probes.
- **results** – Stores outcomes of checks (compliant, non-compliant, partial).
- **relationships:** Each probe links to many checks; checks link to controls.

Evidence Management

- **evidence** – Metadata for documents, reports, or files uploaded to **externally hosted MinIO**.
- **evidence_links** – Associates evidence with controls, checks, or remediation tasks.
- **relationships:** One-to-many between controls and evidence; one-to-one between evidence and MinIO objects.

Compliance Scores

- **scores** – Stores compliance percentage or rating for frameworks, controls, and users.
- **metrics** – Aggregates key performance indicators for dashboards and reports.
- **relationships:** Many-to-one between controls and scores.

Audit Logs and Alerts

- **audit_logs** – Tracks all system actions (who, when, what, and outcome).
- **alerts** – Represents triggered notifications or risk events from backend processes.
- **relationships:** One-to-many between users and audit logs; one-to-many between alerts and tasks.

4.4 Indexing and Query Optimization

Database performance is maintained through intelligent indexing and query optimization strategies.

Indexing Guidelines

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Primary keys indexed by default (UUIDs).
- Foreign key relationships indexed for joins (user_id, control_id, framework_id).
- Partial indexes for high-volume queries (e.g., status = 'active').
- Text search indexes for evidence names and framework descriptions.
- Composite indexes for multi-column queries (e.g., framework_id + control_id).

Query Optimization

- Prisma's query batching and lazy loading minimize redundant calls.
- Prepared statements used for repetitive queries.
- Periodic query plan analysis performed on production.
- Cache layers added for read-heavy queries on reports and dashboards.

4.5 Data Integrity and Relationships

Data consistency is enforced through schema-level constraints and ORM validation.

Integrity Controls

- All foreign keys enforce referential integrity.
- Cascading deletes disabled for critical records; replaced by soft deletes.
- ENUM columns used for fixed value sets (e.g., status, severity).
- Unique indexes ensure data consistency across critical attributes (e.g., framework version).
- Automatic timestamps for every record using created_at and updated_at columns.

Relationship Model Summary

- **users** → **roles** – many-to-one
- **frameworks** → **controls** – one-to-many
- **controls** → **checks** – one-to-many
- **controls** → **evidence** – one-to-many
- **tasks** → **evidence** – many-to-one

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **users** → **audit_logs** – one-to-many

4.6 Backup, Recovery, and Retention

Backup and recovery policies ensure business continuity and data durability.

Backup Policy

- Full database backups performed nightly; differential backups hourly.
- Backup retention for 90 days, extendable for compliance requirements.
- Snapshots encrypted using AES-256 before storage.
- Backups stored on dedicated, access-controlled cloud storage.

Recovery Strategy

- Point-in-time recovery enabled via WAL (Write-Ahead Logging).
- Automated restore testing conducted monthly to validate recovery integrity.
- Disaster recovery environment preconfigured with replicated Postgres instance.

Data Retention

- Evidence and audit data retained for configurable periods with a default of 36 months, satisfying the 400-day regulatory minimum while allowing jurisdiction-specific extensions.
- Automatic archival to cold storage for older records, with immutable copies preserved for at least 7 years.
- GDPR-compliant deletion workflows for user and evidence data.

4.7 Encryption and Security Controls

The database design incorporates multiple layers of security to ensure compliance and protect sensitive data.

Encryption

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Data at rest encrypted using **AES-256**.
- Data in transit encrypted with **TLS 1.3** between API servers and database.
- Prisma-managed secure connection pools for authenticated sessions.

Access Controls

- Role-based access model (RBAC) aligned with Casbin policies.
- Database roles segmented by environment (read-only, admin, service).
- Secrets managed via environment variables and cloud secret vaults.

Audit and Monitoring

- All connections logged with user identity, source IP, and timestamp.
- Database audit logs integrated with backend observability pipelines.
- Alerts configured for unauthorized access attempts or schema modifications.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

5. DevOps & Infrastructure

TL;DR

This section defines the DevOps and infrastructure setup for the AI Governance Platform.

It explains how environments are structured, containerization is managed, CI/CD pipelines are implemented, and infrastructure is provisioned using Infrastructure-as-Code (IaC).

The goal is to ensure a reliable, repeatable, and secure deployment process across all environments — from local development to production — using modern DevOps best practices.

CONFIDENTIAL

5.1 Purpose and Overview

The DevOps and infrastructure layer ensures the platform is deployed, monitored, and maintained in a consistent and secure manner.

It standardizes how environments are built, how containers are deployed, and how changes flow from development to production.

The infrastructure is **cloud-agnostic**, deployable on **AWS or Azure**, and built using **Docker**, **Kubernetes**, and **Terraform**.

Automation ensures minimal manual intervention, enabling rapid iteration and operational resilience.

5.2 Environment Setup

The platform operates across three standard environments:

Development

- Local setup using Docker Compose.
- Connects to externally hosted PostgreSQL and MinIO services.

© 2025 Technical Specifications - Innovista. All rights reserved.

- Hot reloading enabled via Vite and Nodemon.
- Debug logging active for Express backend and React frontend.

Staging

- Cloud-hosted replica of production used for QA, testing, and pre-release validation.
- Automatic builds and deployments triggered via CI/CD pipelines.
- Uses staging credentials, API keys, and sandbox integrations.

Production

- Multi-region, high-availability environment.
- Externally hosted PostgreSQL and MinIO instances configured with encryption and backups.
- Blue-green deployment strategy for zero-downtime releases.
- Metrics and logging integrated with centralized monitoring systems.

Configuration Standards

- Environment-specific .env files stored securely.
- Secrets and credentials managed via cloud vaults (AWS Secrets Manager, Azure Key Vault).
- Environment variables injected dynamically at runtime.

5.3 Repository and Folder Structure

The repository includes a dedicated **infra/** directory at the root level to centralize all infrastructure, automation, and IaC resources.

This folder ensures consistency, reproducibility, and traceability across all environments.

Root-Level Infrastructure Folder

Path	*Purpose*
infra/terraform/	Terraform configuration files and modules for provisioning compute, networking, databases, and storage.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Path	Purpose
infra/scripts/	Shell or Node scripts for automation (deployment, migrations, cleanup).
infra/kubernetes/	Kubernetes manifests or Helm charts for services, pods, and ingress configuration.
infra/monitoring/	Prometheus, Grafana, and Alertmanager configurations for metrics and alerts.
infra/policies/	Policy-as-Code definitions (OPA, Sentinel) for compliance validation in CI/CD.
infra/backups/	Backup scripts and retention definitions for PostgreSQL and MinIO data.

Integration Notes

- The infra/ folder is **version-controlled** and maintained alongside the app code.
- Sensitive data (e.g., .tfvars, credentials) are excluded from version control.
- Terraform state files are stored remotely in encrypted backends (e.g., S3, Terraform Cloud).
- All scripts and manifests are designed to be reusable across **development**, **staging**, and **production**.

5.4 Containerization and Orchestration

All services are containerized using **Docker** to maintain consistency across environments.

Containerization Strategy

- Separate containers for **client**, **server**, and **worker** processes.
- Base images optimized for speed and security.
- Shared internal network for inter-service communication.
- Development uses mounted volumes; production uses immutable images.

Orchestration

- Containers orchestrated using **Kubernetes (K8s)** for production or **Docker Compose** for local development.
- Pods represent independent microservices (frontend, backend, cron, proxy).

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- CPU and memory limits enforced per pod.
- Health checks configured for liveness and readiness.
- Horizontal Pod Autoscaling (HPA) configured based on metrics.

Networking and Logging

- All services communicate via HTTPS and internal DNS within Kubernetes.
- Logs are shipped to centralized monitoring (Grafana Loki, CloudWatch, or ELK).
- Ingress controllers manage routing and TLS termination.

5.5 Continuous Integration and Delivery (CI/CD)

Objectives

- Fully automated pipeline for testing, building, and deploying.
- Reduced manual operations with version-controlled infrastructure.
- Deployment gates for quality and compliance checks.

CI Pipeline

- Triggered on pull requests or merges into main branches.
- Runs linting, unit tests, and build validation.
- Generates versioned Docker images pushed to private registries.

CD Pipeline

- Deploys automatically to **staging**; manual approval for **production**.
- Blue-green or rolling updates minimize downtime.
- Integration and smoke tests post-deployment.

Toolchain

- **GitHub Actions** or **GitLab CI** for automation.
- **Terraform Cloud** for infrastructure provisioning.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **Docker Hub / ECR / ACR** as image registries.
- Slack or email notifications for deployment outcomes.

5.6 Infrastructure as Code (IaC)

Infrastructure is managed using **Terraform**, following modular, declarative, and version-controlled practices.

Core IaC Principles

- Infrastructure defined as code to ensure repeatability.
- Remote encrypted backend for state management.
- Modular design for compute, networking, and database components.
- Code reviews and policy checks before apply.

Terraform Resources

- VPCs, subnets, firewalls, and load balancers.
- Compute clusters (Kubernetes or ECS).
- PostgreSQL and MinIO provisioning.
- IAM roles, service accounts, and encryption policies.
- Monitoring resources and scaling policies.

Approval and Governance

- Pull-request workflow for Terraform changes.
- Policy enforcement via **OPA** or **Terraform Sentinel**.
- Manual approval for production infrastructure updates.

5.7 Monitoring and Observability

Comprehensive observability ensures proactive detection and analysis of system health.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Monitoring Tools

- **Prometheus / CloudWatch** for metrics collection.
- **Grafana** for dashboards and visualization.
- **Alertmanager / PagerDuty** for alert routing.
- **ELK or Loki stack** for centralized logging.

Metrics Tracked

- API latency and throughput.
- Database performance (query duration, connection pool).
- Container resource usage and uptime.
- Framework execution and compliance check frequency.
- SLA target: 99.9% uptime for production workloads.

5.8 Secrets and Configuration Management

Secret Management

- Centralized secret storage using **Vault**, **AWS Secrets Manager**, or **Azure Key Vault**.
- Secrets injected at runtime, never baked into container images.
- Automated rotation policies for keys and tokens.

Configuration Management

- Environment variables defined per environment in .env files.
- Configuration validated in CI before build.
- Sensitive configs restricted to authorized maintainers.
- Role-based access applied at CI/CD and runtime levels.

5.9 Disaster Recovery and Scaling Strategy

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Disaster Recovery (DR)

- Multi-region failover for PostgreSQL and MinIO.
- Automated replication and backup verification.
- Failover scripts maintained in infra/backups/.
- Monthly disaster recovery drills for validation.

Scaling Strategy

- **Horizontal Scaling:** Add containers/pods based on resource metrics.
- **Vertical Scaling:** Adjust compute resources for specific workloads.
- **Database Scaling:** Read replicas for query optimization; connection pooling via Prisma.
- **Storage Scaling:** MinIO bucket lifecycle management and object versioning.

The **DevOps and Infrastructure layer**, centered around the infra/ directory, ensures the platform remains secure, scalable, and reproducible — enabling continuous delivery and operational resilience across all environments.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

6. Security Implementation

TL;DR

This section defines the security implementation across all layers of the AI Governance Platform.

It explains how authentication, authorization, encryption, and auditing are enforced throughout the system.

Built on **Express.js (JavaScript)** and **React.js**, the platform follows a **security-by-design** approach using **JWT**, **Casbin**, **AES-256**, and **TLS 1.3**.

The goal is to maintain data confidentiality, integrity, and availability while ensuring compliance with global security standards such as ISO 27001, SOC 2, and NIST CSF.

CONFIDENTIAL

6.1 Purpose and Overview

The platform is designed with **security by default** as a foundational principle.

Every module — from data collection to reporting — adheres to standardized security controls that protect sensitive information and ensure compliance with industry frameworks.

Security is integrated across:

- **Application Layer** – Authentication, session management, and RBAC.
- **Data Layer** – Encryption, integrity validation, and retention controls.
- **Infrastructure Layer** – Network isolation, TLS encryption, and monitoring.

Security implementation is continuous, automated, and auditable, ensuring that every interaction within the platform remains traceable and compliant.

6.2 Authentication and Authorization

© 2025 Technical Specifications - Innovista. All rights reserved.

Authentication and authorization are implemented using **JWT (JSON Web Tokens)** and **Casbin**, ensuring stateless and policy-based access control.

Authentication

- Users authenticate through a secure login endpoint managed by the **Auth Service**.
- Successful authentication issues a **signed JWT** containing user ID, role, and expiration.
- Tokens are validated on every protected route via middleware.
- Refresh tokens used to extend sessions securely without re-login.
- Expired tokens are invalidated and logged for auditing.

Authorization

- **Casbin** enforces access policies dynamically at runtime.
- Policies define which users or roles can perform actions on specific resources.
- Middleware evaluates user permissions before processing any API request.
- Supports fine-grained, context-aware rules (e.g., per framework, per control, per resource type).

Security Practices

- Passwords hashed with **bcrypt (minimum 12 salt rounds)**.
- Login attempts rate-limited and logged.
- MFA (Multi-Factor Authentication) optional for admin roles.

6.3 Role-Based Access Control (RBAC)

RBAC is the foundation for access management in both frontend and backend systems.

It ensures that users only have permissions necessary to perform their assigned duties.

Role Definitions

- **Admin:** Full system privileges, including configuration and policy management.
- **Compliance Officer:** Manage frameworks, review controls, and approve evidence.
- **Engineer:** Configure probes, submit evidence, and handle remediation.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **Auditor:** View compliance dashboards, scorecards, and audit trails (read-only).
- **System Service:** Internal automation tasks (non-user, machine account).

Implementation

- **Casbin is the mandated RBAC engine** for all services; alternative authorization libraries are not permitted without a formal architecture review and steering committee approval.
- Casbin policies stored in the **PostgreSQL** database.
- RBAC checks performed at middleware level in Express.js.
- Policies cached for performance and invalidated on change events.
- Policy enforcement tested automatically as part of CI pipeline.

6.4 Encryption and Secure Communication

The platform applies **end-to-end encryption** across all data in transit and at rest.

All encryption standards follow current best practices as defined by NIST and ISO.

Data at Rest

- Encrypted using **AES-256** through managed database and storage configurations.
- Prisma ensures ORM-level data validation before persistence.
- MinIO storage encrypted with server-side encryption and KMS-managed keys.
- Secrets stored securely in **Vault / AWS Secrets Manager / Azure Key Vault**.

Data in Transit

- All communication between frontend, backend, and external services occurs over **HTTPS (TLS 1.3)**.
- API endpoints enforce SSL-only connections.
- HSTS (HTTP Strict Transport Security) enabled to prevent downgrade attacks.
- JWT tokens signed using **HMAC SHA-512** with rotating secret keys.

Key Management

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Key rotation scheduled every 90 days.
- Keys stored outside of application runtime in secure vaults.
- Access to keys restricted to DevOps leads and CI/CD systems only.

6.5 Audit Logging and Immutability

The audit logging subsystem ensures that all critical actions are captured, time-stamped, and immutable.

Scope of Audit Logs

- User authentication and access attempts.
- CRUD operations on frameworks, controls, and evidence.
- Administrative actions (role changes, configurations, policy updates).
- System events (service restarts, deployments, incident alerts).

Storage and Integrity

- Logs written using **Winston** and stored as structured JSON.
- Forwarded to centralized log management (ELK, Loki, or CloudWatch).
- Immutable logs protected by **append-only** policies.
- Cryptographic hashing ensures integrity and non-repudiation.

Retention and Access

- Logs retained for a minimum of 36 months by default, exceeding the 400-day regulatory floor and configurable for longer jurisdictional requirements.
- Read access restricted to Admins and Auditors only.
- Archived logs stored in encrypted cold storage for regulatory audits, with immutable snapshots preserved for at least 7 years.

6.6 Security Headers and API Hardening

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

The Express.js server applies modern security headers and middleware to prevent common web vulnerabilities.

Implemented Headers

- **Content-Security-Policy (CSP):** Prevents inline script execution and data injection.
- **X-Frame-Options:** Blocks clickjacking attacks.
- **X-Content-Type-Options:** Prevents MIME-type sniffing.
- **Strict-Transport-Security (HSTS):** Enforces HTTPS-only access.
- **Referrer-Policy:** Restricts sensitive data leakage via headers.
- **Permissions-Policy:** Controls access to browser features (camera, mic, geolocation).

API Hardening

- All input validated and sanitized before processing.
- Rate limiting enforced on login and critical APIs.
- CORS policies restrict origins to trusted domains.
- Hidden endpoints protected behind admin-only access controls.
- Static assets served via secure CDN with cache invalidation.

6.7 Rate Limiting and DDoS Protection

Rate Limiting

- Implemented using Express middleware.
- Request thresholds based on endpoint criticality (e.g., /auth/login limited to 5/minute).
- IP-based throttling with exponential backoff.
- Violations logged and flagged for monitoring.

DDoS and Abuse Prevention

- Web Application Firewall (WAF) configured at ingress layer.
- Load balancer enforces connection throttling for large bursts.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Application-level caching minimizes redundant requests.
- Automated blocking of abusive IPs through monitoring tools (Fail2Ban or Cloudflare).
- Bot detection using user-agent analysis and behavioral heuristics.

6.8 Vulnerability Management and Penetration Testing

The platform follows a proactive approach to vulnerability management, aligned with ISO 27001 and SOC 2 Type II standards.

Vulnerability Management

- Dependencies scanned continuously using **npm audit**, **Snyk**, or **Dependabot**.
- Critical vulnerabilities patched within 48 hours.
- Regular internal scans for configuration drift and secret exposure.
- Security advisories reviewed weekly during sprint planning.

Penetration Testing

- Conducted bi-annually by independent security auditors.
- Includes black-box and white-box testing of APIs and infrastructure.
- Findings triaged using severity scoring (CVSS v3).
- Results tracked in the platform’s risk register for follow-up and remediation.

Continuous Improvement

- Security posture reviewed after every major release.
- Incident response drills conducted quarterly.
- Bug bounty program planned post-production to engage ethical researchers.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

7. Integration Architecture

TL;DR

This section defines how the AI Governance Platform integrates with external systems and data sources.

It describes the structure, workflow, and standards for building connectors, probe integrations, webhooks, and partner APIs.

The integration model enables interoperability between enterprise tools such as **ServiceNow**, **Jira**, **OneTrust**, and **Slack**, ensuring seamless data exchange and automation across governance workflows.

The objective is to provide a secure, extensible framework for external connectivity while maintaining compliance, auditability, and data integrity.

CONFIDE

7.1 Purpose and Overview

The integration layer extends the platform’s functionality by connecting it with third-party tools and enterprise systems.

It enables automated evidence collection, task management, incident reporting, and communication through secure APIs and webhooks.

All integrations follow a **modular and standardized interface design**, ensuring consistency, auditability, and data lineage tracking across systems.

The integration layer adheres to the same **JavaScript-only implementation standard**, maintaining uniform development practices across all services.

7.2 Integration Framework

© 2025 Technical Specifications - Innovista. All rights reserved.

The integration architecture is based on a **modular, pluggable framework** that allows new integrations to be added without altering the core platform.

Key Components

- **Integration Manager:** Central module responsible for registering, configuring, and monitoring integrations.
- **Integration Registry:** Stores metadata about active integrations (type, endpoint, authentication method).
- **Integration Adapters:** Implement data transformation logic between internal models and external APIs.
- **Integration Scheduler:** Handles periodic synchronization and automated data pulls.
- **Event Dispatcher:** Routes inbound and outbound events to appropriate integrations using secure channels.

Framework Design Principles

- Loose coupling through standard REST or message-based APIs.
- Declarative configuration using environment variables and JSON templates.
- Audit logs for all external requests and responses.
- Isolation of integrations in their own sandboxed processes.

7.3 Probe SDK and Data Collection

The **Probe SDK** allows engineers to develop lightweight data collectors (“probes”) that gather compliance evidence from various environments.

Probes can run as background services or on-demand scripts within enterprise systems.

SDK Capabilities

- Simple API to send structured data to the Governance Engine.
- Built-in support for authentication and signing requests with API keys.
- Automatic retries and error logging for resilience.
- Versioning system for backward compatibility.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Typical Use Cases

- Collecting configuration data from model registries or data lakes.
- Extracting audit logs from infrastructure tools.
- Verifying deployment settings, access controls, or policy adherence.
- Running scheduled compliance checks against system APIs.

Data Flow

1. Probe gathers evidence from an enterprise system.
2. Evidence is serialized into JSON and transmitted to the backend through the Probe SDK.
3. The backend validates, stores, and associates the data with relevant controls or frameworks.
4. The result appears in the compliance dashboard and evidence repository.

7.4 External System Integrations

The platform supports predefined integrations with major enterprise systems to automate workflows and data synchronization.

ServiceNow

- Bi-directional integration for compliance task tracking and incident management.
- Automatically creates ServiceNow tickets for failed controls or observations.
- Updates ticket status in real-time based on remediation progress.
- Uses OAuth 2.0 for secure API access.

Jira

- Synchronizes governance tasks and remediation actions as Jira issues.
- Supports automatic issue creation and closure when controls change state.
- Maps control IDs and evidence references to Jira project metadata.
- Includes configuration options for Jira Cloud and self-hosted instances.

OneTrust

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Imports and aligns policy, consent, and risk data from OneTrust’s governance platform.
- Exports AI compliance reports to OneTrust for inclusion in enterprise risk dashboards.
- Enables unified reporting across AI governance and privacy domains.

Slack

- Delivers real-time notifications and compliance alerts to dedicated Slack channels.
- Enables approval workflows and evidence reviews directly from Slack.
- Uses Slack Web API for message delivery with JWT-based authentication.

Each integration is isolated, configurable, and independently deployable, ensuring fault tolerance and versioned control.

7.5 Webhooks and Event Architecture

Webhooks and event subscriptions enable real-time data synchronization between the platform and external systems.

Outbound Webhooks

- Triggered by platform events (e.g., failed check, control updated, task assigned).
- Send structured JSON payloads to external endpoints.
- Support delivery retries with exponential backoff.
- Event delivery logged for traceability and debugging.

Inbound Webhooks

- Allow external systems to send events or updates into the platform.
- Validate signatures and payloads to ensure authenticity.
- Support dynamic registration through the API or admin console.

Event Processing

- Event queues managed via message broker for scalability and fault tolerance.
- Events processed asynchronously to avoid API latency.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Failed webhook attempts logged and retried automatically.

7.6 Partner API and Data Exchange

The **Partner API** enables external systems, auditors, and regulators to programmatically interact with the platform.

It provides a secure interface for reading and writing compliance data while maintaining strict access controls.

Partner API Features

- RESTful API with role-based access control (RBAC) applied via Casbin.
- Supports data export for scorecards, audit logs, frameworks, and controls.
- Enables automated ingestion of evidence or compliance status from third-party tools.
- All endpoints documented through Swagger (OpenAPI 3.0).

Data Governance

- Data sharing restricted to authorized partners and validated tenants.
- API access tokens scoped to least privilege principles.
- All partner transactions logged and timestamped for auditability.

Compliance Alignment

- Partner API adheres to GDPR, ISO 27001, and SOC 2 requirements.
- Supports multi-region data residency enforcement.
- Periodic audits verify data sharing and retention policies.

7.7 Authentication, Security, and Governance

Security is enforced across all integrations to ensure confidentiality, integrity, and availability of exchanged data.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Authentication

- Integrations authenticated via OAuth 2.0, API keys, or signed JWTs.
- API keys generated and rotated through the admin console.
- Service-to-service communication secured through mutual TLS (mTLS).

Authorization

- Casbin policies enforce per-integration access control.
- Each integration registered with unique identifiers and scopes.
- API endpoints protected by JWT middleware and signed payload validation.

Governance and Auditability

- Every integration interaction logged in the audit system.
- Integration health monitored via periodic heartbeat checks.
- Alerts generated for API failures, expired tokens, or rate-limit violations.
- All external data exchanges subject to encryption, versioning, and approval workflows.

The integration architecture provides a scalable and secure foundation for connecting the AI Governance Platform with enterprise systems — enabling continuous compliance automation across diverse technology ecosystems.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

8. Deployment & Environment Guide

TL;DR

This section provides detailed instructions for setting up, configuring, and deploying the AI Governance Platform across all environments — local, staging, and production.

It covers environment variables, build and run commands, seeding and migrations, deployment checklists, and cloud configuration standards.

The goal is to ensure consistent, reliable, and secure deployments regardless of environment or hosting provider.

8.1 Purpose and Overview

This guide outlines how to set up and deploy the platform in different environments.

It standardizes the deployment process to minimize errors and ensure reproducibility across local, staging, and production systems.

The platform can be deployed on **AWS, Azure, or GCP**, using **Docker containers**, **Kubernetes clusters**, and **Terraform-managed infrastructure**.

All configurations and build scripts reside within the **infra/** directory for consistency.

8.2 Environment Types

Local Development

- Runs via Docker Compose or individual npm commands.
- Connects to external PostgreSQL and MinIO services.
- Uses local .env configuration.
- Enables live reloading for frontend (Vite) and backend (Nodemon).

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Debug logging enabled by default.

Staging

- Mirrors the production environment for QA and UAT testing.
- Automatically built and deployed through CI/CD pipelines.
- Connects to staging instances of external services (PostgreSQL, MinIO, email).
- Protected by basic authentication or IP whitelisting.
- Includes synthetic test data for validation.

Production

- Deployed on a cloud-managed infrastructure (Kubernetes or ECS).
- Uses external, high-availability PostgreSQL and MinIO instances.
- Follows blue-green or rolling deployment strategy.
- Logging and monitoring integrated with centralized observability systems.
- HTTPS and TLS enforced at all entry points.

8.3 Local Development Setup

Prerequisites

- Node.js LTS (v20 or above)
- Docker and Docker Compose
- PostgreSQL (remote or local instance)
- MinIO (external or local container)
- npm or yarn

Setup Steps

1. Clone the repository.
2. Copy `.env.example` to `.env` and update environment variables.
3. Run `npm install` in both `/client` and `/server` directories.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

4. Start backend and frontend using Docker Compose or separate terminals:

- Backend: npm run dev (from /server)
- Frontend: npm run dev (from /client)

5. Access the frontend at http://localhost:5173 (Vite default).

6. Confirm API connectivity at http://localhost:4000/api/health.

Development Tips

- Backend and frontend code auto-reload on file changes.
- Debug logs available in terminal output.
- Use .env.dev for local overrides not committed to Git.

8.4 Environment Variables and Configuration

The platform uses a unified .env file pattern for managing environment configurations.

Configuration Principles

- Sensitive credentials stored in vaults (Vault, AWS Secrets Manager).
- .env.example acts as a template for environment variables.
- Environment-specific files: .env.dev, .env.staging, .env.prod.
- Environment variables injected automatically via CI/CD during build.

Core Environment Variables

Variable	*Description*
PORT	API port for Express server.
VITE_API_URL	Backend API endpoint for frontend.
DATABASE_URL	Connection string for external PostgreSQL.
MINIO_ENDPOINT	External MinIO endpoint URL.
MINIO_ACCESS_KEY	Access key for MinIO.
MINIO_SECRET_KEY	Secret key for MinIO.
JWT_SECRET	JWT signing key for authentication.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Variable	*Description*
EMAIL_HOST	SMTP host for Nodemailer.
EMAIL_USER	SMTP username.
EMAIL_PASS	SMTP password.
NODE_ENV	Runtime environment (development, staging, production).

8.5 Database Seeding and Migrations

Database schema management is handled by **Prisma ORM**, ensuring consistency across environments.

Migrations

- Run automatically during CI/CD or manually with npx prisma migrate deploy.
- Migrations stored under server/prisma/migrations/.
- Schema validated against production database before deployment.

Seeding

- Seeds create base data such as roles, admin users, and framework templates.
- Run npx prisma db seed after initial migration.
- Seeding scripts reside in server/prisma/seed.js.
- Separate seeds for development and staging environments.

Rollbacks

- Use npx prisma migrate resolve --rolled-back for controlled rollbacks.
- Migrations versioned and reviewed via pull requests.

8.6 Build and Deployment Process

The platform is containerized and deployed using CI/CD pipelines defined under the **infra/** folder.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

Build Steps

1. CI pipeline triggers build on commit to main or release branch.
2. Backend and frontend Docker images are built separately.
3. Images tagged with version and commit hash (e.g., v1.2.0-commitsha).
4. Images pushed to private container registry (ECR, ACR, or Docker Hub).

Deployment Steps

1. Staging deployment runs automatically after successful build.
2. Manual approval required for production release.
3. Terraform provisions necessary infrastructure if not present.
4. Kubernetes manifests or Helm charts deployed via infra/kubernetes/.
5. Post-deployment health checks validate system readiness.

Rollback Procedure

- Blue-green deployment ensures instant rollback by switching traffic.
- Previous container versions retained for 7 days.
- Rollback initiated through CI/CD dashboard or Kubernetes commands.

8.7 Cloud Deployment Checklist

Pre-Deployment

- [ ] Environment variables validated and secrets verified.
- [ ] Database migrations and seed scripts completed.
- [ ] SSL certificates configured for ingress or load balancer.
- [ ] Logging and metrics targets configured.
- [ ] Monitoring dashboards updated for new services.

Post-Deployment

- [ ] Health check endpoints returning 200 OK.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- [ ] Frontend connected successfully to backend API.
- [ ] Logs and metrics visible in observability platform.
- [ ] Data persistence verified (PostgreSQL and MinIO).
- [ ] Backups initiated successfully.

8.8 Verification and Post-Deployment Steps

Verification Process

- Run automated smoke tests from CI after deployment.
- Verify compliance workflows, dashboards, and evidence uploads.
- Validate email delivery through Nodemailer integration test.
- Confirm Casbin policies are loaded correctly for RBAC.

Post-Deployment Activities

- Tag release in Git and document version in release notes.
- Archive old build artifacts for traceability.
- Rotate JWT and API keys if required.
- Conduct security scan to confirm no new vulnerabilities introduced.

This standardized deployment guide ensures all developers and DevOps engineers follow a unified process, minimizing risk and maintaining operational consistency across all environments.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

9. Testing & QA Framework

TL;DR

This section defines the testing and quality assurance (QA) framework for the AI Governance Platform.

It outlines the testing strategy, tools, workflows, and coverage standards required to ensure reliability, performance, and security across all system components.

The goal is to maintain continuous quality validation throughout development and deployment, ensuring that every build is production-ready.

CONFIDENTIAL

9.1 Purpose and Overview

Testing and QA are core parts of the development lifecycle, ensuring the system remains stable, performant, and secure through automated and manual validation.

The testing framework ensures that each release meets defined quality thresholds before reaching production.

Testing activities are integrated into CI/CD pipelines and cover the entire platform — frontend, backend, database, and integrations.

All tests are written in **JavaScript**, following the project's "no TypeScript" principle for simplicity and uniformity.

9.2 Testing Strategy

The testing strategy follows a **shift-left approach**, embedding testing early in the development cycle.

Automation ensures continuous validation of builds and infrastructure, while manual QA focuses on usability and exploratory testing.

© 2025 Technical Specifications - Innovista. All rights reserved.

Core Objectives

- Prevent regression across releases.
- Validate feature functionality before merge.
- Ensure cross-environment consistency.
- Verify security and compliance features.
- Monitor system performance and reliability under load.

Testing Philosophy

- **Automate first:** Priority given to repeatable tests that can be automated.
- **Test small, test often:** Incremental validation through unit and integration tests.
- **Fail fast:** CI pipelines halt on test failures, preventing broken builds.
- **Quality as code:** Test definitions stored and versioned alongside source code.

9.3 Testing Levels and Scope

Unit Testing

- Tests individual functions, components, and modules.
- Ensures isolated code behavior correctness.
- Mocking used for API calls and external dependencies.
- Frameworks: **Jest** for backend and **Vitest** for frontend.

Integration Testing

- Validates interactions between modules (e.g., backend routes with database).
- Confirms contract integrity between frontend and backend APIs.
- Uses in-memory or sandboxed PostgreSQL instances for test execution.

API Testing

- Verifies all REST endpoints using automated tools like **Postman** and **Newman**.
- Ensures endpoint reliability, response structure, and authentication flows.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Swagger (OpenAPI) used as the contract source for API test generation.

End-to-End (E2E) Testing

- Tests complete user flows through the UI and backend.
- Conducted using **Cypress** for browser automation.
- Includes login, evidence upload, and framework mapping validation.

Security Testing

- Automated vulnerability scans via **OWASP ZAP** or **k6** extensions.
- Tests for injection, authentication flaws, and misconfiguration.
- Integrated into CI pipeline for every major release.

Performance Testing

- Load and stress testing performed using **k6**.
- Monitors system throughput, response time, and resource utilization.
- Validates scaling thresholds defined in infrastructure benchmarks.

9.4 Testing Tools and Frameworks

Layer	*Tool*	*Purpose*
Backend	Jest	Unit and integration testing for Express.js APIs.
Frontend	Vitest	Component and unit testing for React (Vite-based).
E2E/UI	Cypress	End-to-end functional and regression testing.
API	Postman / Newman	API contract and endpoint testing.
Performance	k6	Load, stress, and soak testing.
Security	OWASP ZAP / Snyk	Vulnerability and dependency scanning.
CI/CD Integration	GitHub Actions	Automated execution of test suites during builds.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

9.5 Test Data and Fixtures

Test data management ensures that test runs remain consistent, isolated, and reproducible across environments.

Guidelines

- Use dedicated testing database schemas separate from production.
- Reset test data between runs using migration rollback scripts.
- Fixtures stored under /server/tests/fixtures and /client/tests/mocks.
- Sensitive credentials replaced with anonymized or mock values.
- Synthetic datasets generated for performance and E2E testing.

Data Isolation

- Backend test runs use ephemeral containers or in-memory databases.
- Test evidence uploaded to temporary MinIO buckets with cleanup scripts.
- QA environments refreshed nightly using CI automation.

9.6 Continuous Testing in CI/CD

Testing is fully integrated into CI/CD workflows to enforce continuous quality gates.

Pipeline Integration

- Unit and integration tests run on every pull request.
- E2E and performance tests triggered on staging deployment.
- Security scans executed weekly or on dependency updates.
- Failed test runs block merge or deployment until resolved.

CI/CD Stages

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

1. **Build and Lint** – Verify syntax, linting, and formatting.
2. **Unit Tests** – Execute Jest and Vitest suites.
3. **Integration Tests** – Run API-level and database tests.
4. **E2E Tests** – Launch Cypress tests in headless mode.
5. **Performance Tests** – Run k6 load tests pre-release.
6. **Security Scans** – Automated dependency and vulnerability checks.

Reporting

- Test results summarized in CI logs and stored in build artifacts.
- Reports exported as HTML dashboards or uploaded to test management tools.
- Notifications sent to developers via Slack or email on failure.

9.7 Coverage Metrics and Reporting

Coverage metrics define the minimum acceptable test coverage for all services.

Coverage Targets

- **Unit Tests:** ≥ 85% of code coverage.
- **Integration Tests:** ≥ 70% of API endpoints covered.
- **Frontend Components:** ≥ 80% rendered and tested.
- **E2E Flows:** 100% coverage of critical user journeys.

Reporting

- Code coverage tracked through **Istanbul (nyc)** integrated with Jest and Vitest.
- Reports generated automatically and stored as CI artifacts.
- Coverage thresholds enforced — builds fail if below minimum standards.
- Historical trend analysis maintained through dashboards for visibility.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

9.8 Quality Assurance Processes

Manual QA

- Conducted by QA engineers during pre-release validation.
- Focuses on usability, accessibility, and exploratory testing.
- Regression testing checklist updated with every release cycle.

Acceptance Testing

- Product owners validate features against acceptance criteria.
- QA and developers jointly sign off before release.
- Test results documented in the release notes.

Bug Lifecycle

1. Issue reported via project management tool (e.g., Jira).
2. Triaged by QA lead and prioritized by severity.
3. Fix validated in staging environment.
4. Ticket closed only after successful re-test.

Continuous Improvement

- Weekly QA retrospectives to identify testing gaps.
- Monthly quality reports track metrics like pass rate, MTTR, and defect density.
- Feedback loop integrated into sprint planning for process enhancement.

The Testing & QA Framework ensures the AI Governance Platform maintains high levels of reliability and security through structured, automated, and continuously improving testing practices.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

10. Coding Standards & Governance

TL;DR

This section defines the coding standards, review processes, and governance practices that ensure consistency, maintainability, and security across the AI Governance Platform's codebase.

It establishes rules for naming, branching, commits, documentation, and reviews to maintain high-quality engineering discipline.

The goal is to align all contributors under a unified development culture that supports scalability, clarity, and compliance readiness.

CONFIDENTIAL

10.1 Purpose and Overview

Coding standards ensure that all developers produce readable, secure, and consistent JavaScript code across the frontend, backend, and shared libraries.

They enforce a shared baseline of quality and maintainability that supports scalability, long-term collaboration, and compliance audits.

The entire codebase adheres to **JavaScript-only implementation** (no TypeScript), emphasizing simplicity and uniform syntax throughout the platform.

10.2 Code Style and Formatting

The project enforces consistent style through automated tools and pre-commit checks.

Tools

- **ESLint:** Enforces syntax, structure, and coding best practices.
- **Prettier:** Handles code formatting automatically for readability.

© 2025 Technical Specifications - Innovista. All rights reserved.

- **Husky:** Runs pre-commit hooks to validate code quality before commits.
- **EditorConfig:** Ensures uniform indentation and file encoding across IDEs.

General Style Rules

- Indentation: 2 spaces.
- Line length: Maximum 100 characters.
- Semicolons: Required.
- Quotation marks: Single quotes for strings.
- Variable naming: camelCase for variables, PascalCase for classes.
- Imports grouped logically: external libraries → internal modules → utilities.
- Avoid unused imports and variables.
- Prefer const over let for immutability.

Automation

- npm run lint checks code quality before CI pipeline build.
- Commits blocked if lint or formatting errors exist.
- Prettier runs automatically via Husky on staged files.

10.3 Naming Conventions

Consistency in naming helps improve discoverability and code readability across the entire repository.

Directories and Files

- All directories and files named in **kebab-case** (e.g., user-service.js, framework-controller.js).
- Tests mirror source file structure (<filename>.test.js).
- Configuration and environment files follow standard naming (.env, .env.example).

Classes and Functions

- Classes use **PascalCase** (e.g., UserController, AuthService).

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Functions and variables use **camelCase**.
- Constants written in **UPPER*SNAKE*CASE** (e.g., MAX_UPLOAD_SIZE).
- React components always in **PascalCase**.

Commits and Branches

- Branch names follow convention:

feature/<name>, bugfix/<name>, hotfix/<name>, chore/<name>.

- Commit messages use lowercase imperative tone:

add login validation, fix auth token refresh bug, update linting rules.

10.4 Branching and Version Control Policy

Version control follows **Git Flow**, providing a structured release process and controlled collaboration.

Branch Structure

- **main:** Stable production-ready code.
- **develop:** Integration branch for upcoming releases.
- **feature/** branches: Active development of new features.
- **bugfix/** branches: For minor fixes before release.
- **hotfix/** branches: Emergency fixes applied directly to production.

Workflow

1. Create a feature branch from develop.
2. Submit a pull request (PR) once work is complete.
3. PR reviewed, tested, and merged into develop.
4. develop merged into main during release with version tagging.

Versioning

- Semantic Versioning (MAJOR.MINOR.PATCH).

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Example: v1.3.2 = backward-compatible patch after minor update.
- Version tags automatically generated by CI on successful release builds.

10.5 Commit Message Standards

Commit messages document intent and simplify traceability for code changes.

Format

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

11. Future Extensions

TL;DR

This section outlines the planned future extensions and long-term roadmap for the AI Governance Platform.

It identifies potential areas for modular growth, marketplace integration, AI-driven analytics, and multi-region architecture.

The objective is to ensure that the current system design remains scalable, forward-compatible, and adaptable to emerging technologies and governance requirements.

CONFIDENTIAL

11.1 Purpose and Vision

The future architecture of the AI Governance Platform focuses on **sustainability, intelligence, and extensibility**.

As regulatory requirements, data volumes, and AI models evolve, the system must support new frameworks, integrations, and analytics use cases.

This section defines how the platform can evolve into a **multi-tenant, globally distributed, and AI-assisted governance system**, capable of continuous compliance at scale.

11.2 Modular Microservice Evolution

The monolithic Express.js backend can evolve into a **microservice-based architecture** while maintaining compatibility with existing APIs.

Transition Objectives

- Decouple business logic modules (Auth, Governance, Evidence, Frameworks, Notifications).
- Containerize and independently scale each service.

© 2025 Technical Specifications - Innovista. All rights reserved.

- Use a shared message broker (e.g., NATS or Kafka) for inter-service communication.
- Deploy microservices via Kubernetes Helm charts in separate namespaces.
- Introduce API Gateway for unified access control and traffic management.

Benefits

- Horizontal scalability per service type.
- Independent deployment cycles.
- Fault isolation and improved resilience.
- Easier feature rollout and rollback.

Future Enhancements

- Migrate to **GraphQL Gateway** for federated API aggregation.
- Implement distributed tracing and telemetry across services.
- Support hybrid on-premise and cloud deployment models for enterprise clients.

11.3 API Marketplace and Monetization Roadmap

To extend platform adoption, the roadmap includes developing an **API Marketplace** for partners, auditors, and developers.

Marketplace Features

- Centralized portal for discovering, testing, and consuming APIs.
- Self-service API key management and usage analytics.
- Tiered pricing based on data volume, access scope, and SLA.
- Subscription and billing integration with payment gateways.
- Sandbox environment for API prototyping and compliance testing.

Monetization Strategy

- Offer freemium tier with rate limits for open access.
- Premium and enterprise tiers with extended limits and SLA guarantees.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- Revenue sharing model for third-party integration developers.

Technical Foundation

- Managed via API Gateway with request metering and billing hooks.
- Integrated with Partner API (Section 7.6) for consistent authentication.
- Analytics dashboard for API usage and performance tracking.

11.4 AI-Powered Governance Analytics

As data grows, integrating **machine learning and AI-based analytics** will provide predictive and prescriptive governance insights.

Planned Capabilities

- Predict compliance risk based on past audit results and framework maturity.
- Auto-classify evidence and map it to controls using NLP models.
- Generate dynamic compliance scoring using AI-powered inference models.
- Detect anomalies in audit logs or check execution patterns.
- Forecast remediation timelines based on historical task data.

Implementation Roadmap

- Introduce a dedicated **AI Analytics Service** connected to PostgreSQL and MinIO.
- Use open-source frameworks (e.g., TensorFlow.js, PyTorch via microservice API).
- Train models using anonymized and aggregated customer datasets.
- Provide explainable AI (XAI) outputs to ensure audit transparency.

Compliance Alignment

- Adhere to **EU AI Act** and **OECD AI Principles** for responsible AI usage.
- Maintain human oversight for all automated compliance decisions.
- Log all AI-generated outputs for auditability and validation.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

11.5 Multi-Region and Data Localization Strategy

As compliance laws evolve globally, multi-region deployments and data residency control become critical.

Multi-Region Architecture

- Deploy regional Kubernetes clusters for geographic redundancy.
- Use geo-aware load balancers for routing user requests.
- Synchronize metadata between regions through asynchronous replication.
- Deploy PostgreSQL read replicas per region for performance optimization.

Data Localization

- Configurable data residency enforcement (e.g., EU, APAC, North America).
- Region-specific MinIO instances for localized file storage.
- Encryption keys and secrets managed separately per jurisdiction.
- Cross-border data sharing restricted by policy and user consent.

Global Monitoring

- Unified control plane for infrastructure management across all clusters.
- Region-specific monitoring with aggregated global dashboards.
- Compliance checks per jurisdiction integrated into CI/CD pipeline.

11.6 Interoperability and Standards Alignment

The platform will evolve to support open compliance standards and API interoperability.

Planned Standards

- **Open Control Framework (OCF):** Mapping alignment for global controls.
- **OpenAPI Federation:** Consistent schema and documentation exchange.
- **STIX/TAXII:** Integration for security and incident sharing.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

- **JSON-LD / RDF:** Semantic data representation for AI governance metadata.

Interoperability Goals

- Enable third-party tools to integrate without custom development.
- Support import/export of compliance data in standardized formats.
- Promote ecosystem collaboration through public developer documentation.

Community Engagement

- Publish open-source SDKs for Probe and Partner APIs.
- Participate in cross-industry governance consortiums.
- Align roadmap with evolving AI governance and privacy standards.

11.7 Long-Term Scalability and Maintenance

The platform’s architecture is designed for long-term sustainability, maintainability, and operational scalability.

Long-Term Maintenance Principles

- Modular service boundaries with independent ownership.
- Clear versioning and deprecation policies.
- Continuous upgrade of runtime dependencies (Node.js, React, Prisma).
- Automated tests for backward compatibility and migration validation.

Scalability Roadmap

- Adopt serverless compute for transient compliance tasks.
- Integrate distributed caching (Redis, Memcached) for high-performance queries.
- Implement edge computing for low-latency evidence validation.
- Leverage autoscaling and cost-optimization strategies in Kubernetes clusters.

Vision

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.

The platform will evolve into a **global governance ecosystem** — intelligent, distributed, and standards-aligned — empowering organizations to achieve continuous compliance and ethical AI governance at scale.

CONFIDENTIAL

© 2025 Technical Specifications - Innovista. All rights reserved.