RiskFortress: Enterprise Governance, Risk & Compliance Platform

Technical White Paper

Document Version: 2.1 (Technical)
Date: January 2025
Classification: Public
Audience: Technical Architects, Developers, IT Operations

Table of Contents

Executive Summary

RiskFortress is a modern, enterprise-grade GRC platform built on a TypeScript-based full-stack architecture. The system leverages Next.js 15 for the frontend, Fastify for the backend API, and Prisma ORM for database management. This technical white paper provides detailed architectural documentation, implementation details, and technical specifications for developers, architects, and IT operations teams.

Key Technical Highlights:

- **TypeScript-first**: Full type safety across frontend and backend
- **Modern Stack**: Next.js 15 (App Router), Fastify, Prisma, PostgreSQL/SQLite
- **Clean Architecture**: Separation of concerns with service layer pattern
- **Multi-Tenant**: Client-specific database isolation
- **RESTful API**: Comprehensive API with JWT authentication
- **Real-Time**: Live dashboards with TanStack Query
- **AI Integration**: OpenAI and Ollama support for intelligent features

---

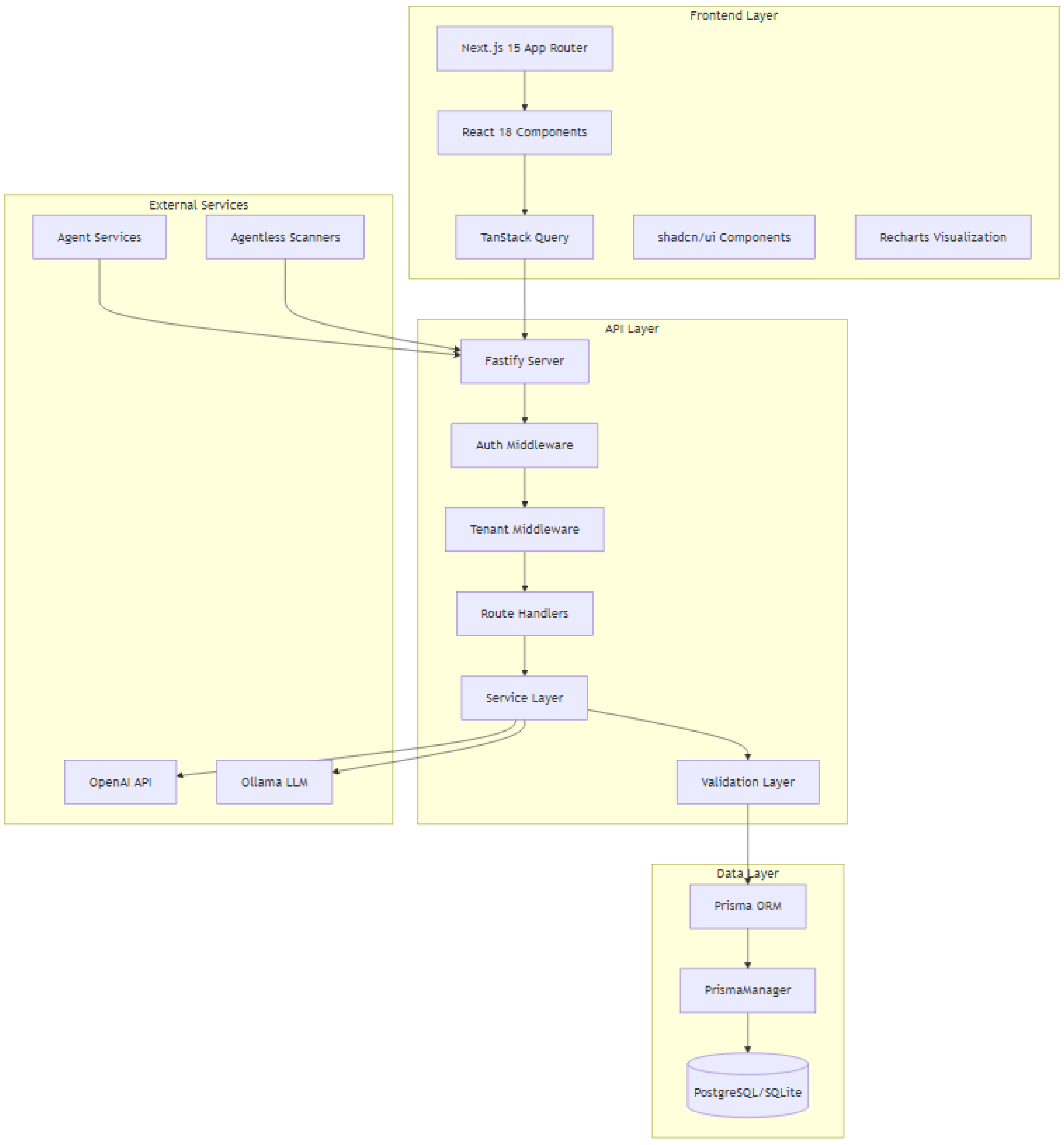
System Architecture

High-Level Architecture

RiskFortress follows a three-tier architecture pattern:

```
┌─────────────────────────────────────────────────────────┐
│                    Presentation Layer                     │
│  Next.js 15 (App Router) + React 18 + TypeScript        │
│  - Server-Side Rendering                                  │
│  - Client-Side State Management                         │
│  - Real-Time Dashboards                                   │
└────────────────────┬────────────────────────────────────┘
                     │ HTTP/REST
┌────────────────────▼────────────────────────────────────┐
│                    Application Layer                      │
│  Fastify API Server + TypeScript                          │
│  - Route Handlers                                         │
│  - Service Layer                                          │
│  - Business Logic                                         │
│  - Authentication & Authorization                         │
└────────────────────┬────────────────────────────────────┘
                     │ Prisma ORM
┌────────────────────▼────────────────────────────────────┐
│                      Data Layer                           │
│  PostgreSQL / SQLite                                      │
│  - Client-Specific Databases                              │
│  - Schema Migrations                                      │
│  - Data Isolation                                         │
└──────────────────────────────────────────────────────────┘
```

Component Architecture



Technical Stack

Frontend Stack

Core Framework:

- **Next.js 15**: React framework with App Router
    - Server Components for performance
    - Client Components for interactivity
    - API Routes for server-side logic
    - Built-in optimization (image, font, script)

UI Libraries:

- **React 18**: Component library with hooks
- **TypeScript**: Type-safe development
- **TailwindCSS**: Utility-first CSS framework
- **shadcn/ui**: High-quality component library built on Radix UI
- **Recharts**: Data visualization library
- **Lucide React**: Icon library

State Management:

- **TanStack Query (React Query)**: Server state management
    - Automatic caching
    - Background refetching
    - Optimistic updates
- **React Hooks**: Local component state
- **URL State**: Query parameters for navigation

Development Tools:

- **ESLint**: Code linting
- **TypeScript**: Type checking
- **PostCSS**: CSS processing

- **Autoprefixer**: CSS vendor prefixing

Backend Stack

Core Framework:

- **Fastify**: High-performance web framework
  - Plugin-based architecture
  - Built-in JSON schema validation
  - TypeScript support
  - Request/response logging

Database:

- **Prisma ORM**: Type-safe database access
  - Schema definition in Prisma Schema Language
  - Automatic migrations
  - Type-safe queries
  - Connection pooling
- **SQLite**: Development database
- **PostgreSQL**: Production database (recommended)

Validation:

- **Zod**: Schema validation library
  - Runtime type checking
  - Type inference
  - Custom error messages

Authentication:

- **@fastify/jwt**: JWT token management
- **bcryptjs**: Password hashing

File Processing:

- **@fastify/multipart**: File upload handling
- **csv-parser**: CSV file parsing

- **exceljs**: Excel file processing

Reporting & Export

- **Puppeteer**: Headless Chrome for PDF generation
- **ExcelJS**: Excel file generation
- **docx**: Word document generation
- **json2csv**: CSV export

AI Integration

- **OpenAI SDK**: GPT-4 and GPT-4o-mini integration
- **Ollama**: Local LLM deployment support
- **Custom AI Service Layer**: Abstraction for multiple providers

---

Database Design

Schema Overview

The database schema is defined using Prisma Schema Language. Key design principles:

1. **Normalization**: Proper relational design with foreign keys
2. **Type Safety**: Prisma generates TypeScript types
3. **Migrations**: Version-controlled schema changes
4. **Indexes**: Optimized queries with strategic indexing

Core Models

User Model

```
model User {
  id        String   @id @default(cuid())
  email     String   @unique
  password  String   // bcrypt hashed
  name      String
  role      UserRole @default(ANALYST)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

Framework Hierarchy

```
model Framework {
  id          String              @id @default(cuid())
  code        String              @unique
  name        String
  description String
  inScope     Boolean             @default(true)
  categories  FrameworkCategory[]
}

model FrameworkCategory {
  id          String             @id @default(cuid())
  name        String
  description String
  frameworkId String
  framework   Framework          @relation(...)
  controls    FrameworkControl[]
}

model FrameworkControl {
  id          String                 @id @default(cuid())
  controlId   String
  title       String
  description String
  questions   Json? // Question definitions
  categoryId  String
  category    FrameworkCategory    @relation(...)
  subcontrols FrameworkSubcontrol[]
}
```

Asset & Mapping Models

```
model Asset {
  id              String                @id @default(cuid())
  name            String
  type            String
  confidentiality Int                   @default(3)
  integrity       Int                   @default(3)
  availability    Int                   @default(3)
  riskRating      String                @default("Medium")
  mappings        AssetControlMapping[]
  assessments     Assessment[]
  risks           RiskItem[]
}

model AssetControlMapping {
  id          String           @id @default(cuid())
  assetId     String
  controlId   String
  maturity    MaturityLevel
  weight      Float            @default(1.0)
  comment     String?
  remediation String?
  asset       Asset            @relation(...)
  control     FrameworkControl @relation(...)
}
```

Assessment Model

```
model Assessment {
  id            String   @id @default(cuid())
  assetId       String
  frameworkId   String
  controlId     String
  subcontrolId String?
  maturity      Int?     // 1-4 scale
  response      String?
  evidence      String?
  status        String   // not_started | in_progress | completed
  answers       AssessmentAnswer[]
}

model AssessmentAnswer {
  id            String   @id @default(cuid())
  assessmentId String
  question      String
  type          String   // boolean | text | multiple-choice
  answer        Json?    // boolean, string, or array
  weight        Float    @default(1.0)
  score         Float?
}
```

Database Relationships

Key Relationships:

- Framework → Categories → Controls → Subcontrols (1:N)
- Asset → Mappings → Controls (N:M via AssetControlMapping)
- Asset → Assessments → Answers (1:N)
- Control → Risks (1:N)
- Agent → ScanResults (1:N)

Indexing Strategy

Strategic indexes for performance:

- Foreign key indexes on all relation fields

- Composite indexes for common query patterns
- Unique constraints on business keys (email, framework code)
- Indexes on frequently filtered fields (status, createdAt)

Multi-Tenancy Implementation

Database Isolation:

- Each client has a dedicated database file (SQLite) or schema (PostgreSQL)
- `PrismaManager` handles database switching
- No cross-client data access possible at the database level

Implementation:

```
class PrismaManager {
  private _client: PrismaClient;
  private _clientId: string | null;

  async switchDatabase(url: string, clientId: string | null) {
    await this._client.$disconnect();
    this._client = new PrismaClient({
      datasources: { db: { url } }
    });
    this._clientId = clientId;
  }
}
```

API Architecture

API Structure

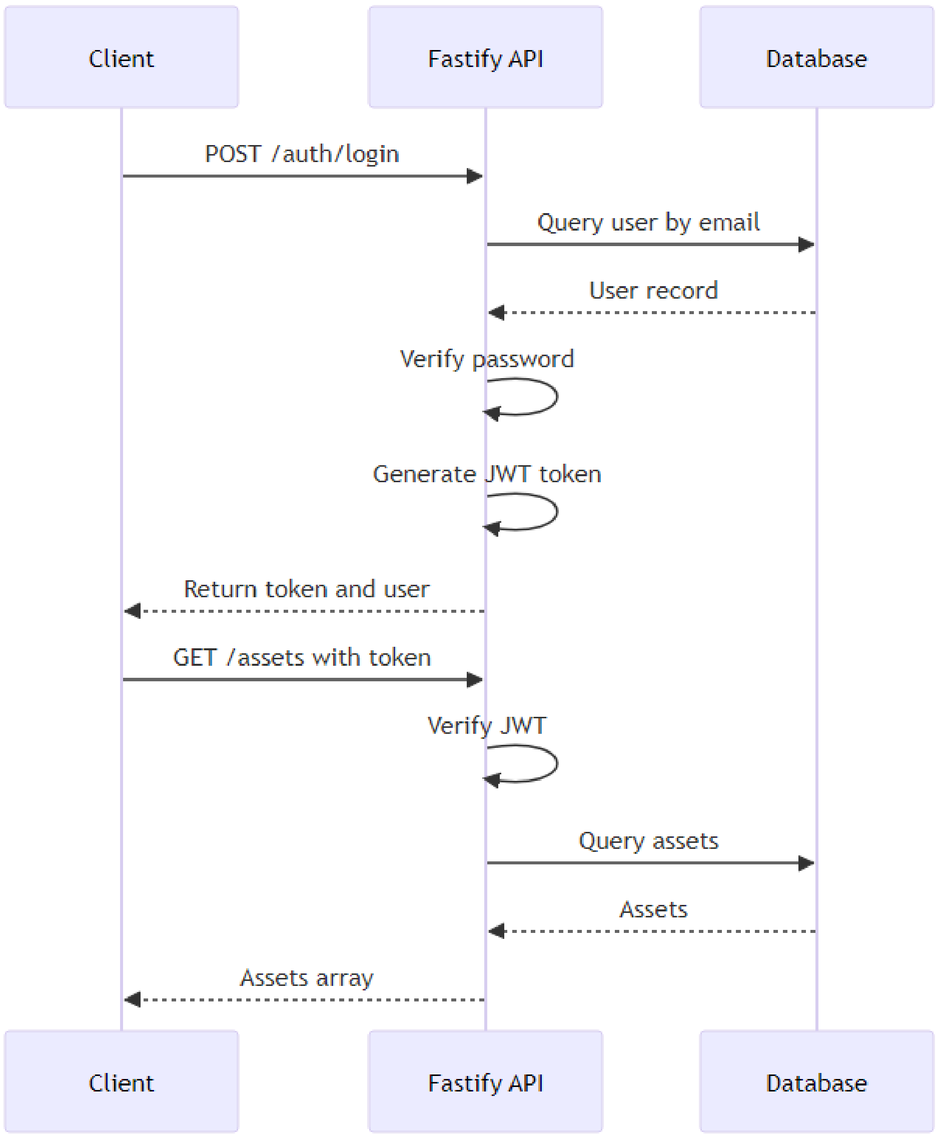
The API is organized into modular route handlers:

```
server/routes/
├── auth.ts            # Authentication endpoints
├── assets.ts          # Asset CRUD
├── frameworks.ts      # Framework management
├── assessment.ts      # Assessment workflows
├── risk.ts            # Risk register
├── reports.ts         # Report generation
├── import.ts          # Data import
├── agents.ts          # Agent management
├── agentless.ts       # Agentless scanning
├── ai.ts              # AI integration
├── settings.ts        # Settings and branding
└── database.ts        # Database management
```

API Design Principles

1. **RESTful Conventions**: Standard HTTP methods and status codes
2. **Consistent Response Format**: Standardized error and success responses
3. **Authentication**: JWT-based authentication on protected routes
4. **Validation**: Zod schema validation for all inputs
5. **Error Handling**: Consistent error responses with codes

Authentication Flow



API Endpoints

Authentication:

- `POST /auth/login` - User login
- `POST /auth/register` - User registration

Assets:

- `GET /assets?page=1&pageSize=20&search=...` - List assets (paginated)

- `POST /assets` - Create asset
- `PUT /assets/:id` - Update asset
- `DELETE /assets/:id` - Delete asset

Frameworks:

- `GET /frameworks` - List frameworks
- `GET /frameworks/:id` - Get framework with categories and controls
- `POST /frameworks` - Create framework
- `PUT /frameworks/:id` - Update framework
- `PUT /frameworks/:id/scope` - Update framework scope

Assessments:

- `POST /assessment` - Create assessment
- `GET /assessment/:assetId` - Get assessments for asset
- `PUT /assessment/:id` - Update assessment with answers
- `GET /assessment/:id/scores` - Get calculated scores

Risk Register:

- `GET /risk` - List risks
- `POST /risk` - Create risk
- `PUT /risk/:id` - Update risk
- `DELETE /risk/:id` - Delete risk

Reports:

- `POST /reports/html` - Generate HTML report
- `POST /reports/pdf` - Generate PDF report
- `POST /reports/docx` - Generate Word document

Import:

- `POST /api/import/security` - Import Prowler/Wazuh scan
- `POST /api/import/dataset` - Import custom dataset
- `GET /api/import/preview` - Preview import data

Request/Response Examples

Create Asset:

```
// Request
POST /assets
{
  "name": "Production Server 01",
  "type": "server",
  "owner": "IT Operations",
  "confidentiality": 3,
  "integrity": 3,
  "availability": 3,
  "riskRating": "Medium"
}

// Response
{
  "id": "clx123...",
  "name": "Production Server 01",
  "type": "server",
  "createdAt": "2024-12-01T10:00:00Z"
}
```

Create Assessment:

```
// Request
POST /assessment
{
  "assetId": "clx123...",
  "frameworkId": "clx456...",
  "controlId": "clx789...",
  "maturity": 3,
  "response": "Control is implemented with monitoring",
  "evidence": "Link to documentation"
}

// Response
{
  "id": "clxabc...",
  "status": "completed",
  "scores": {
    "questionScore": 0.85,
    "controlScore": 0.75,
    "categoryScore": 0.70
  }
}
```

Frontend Architecture

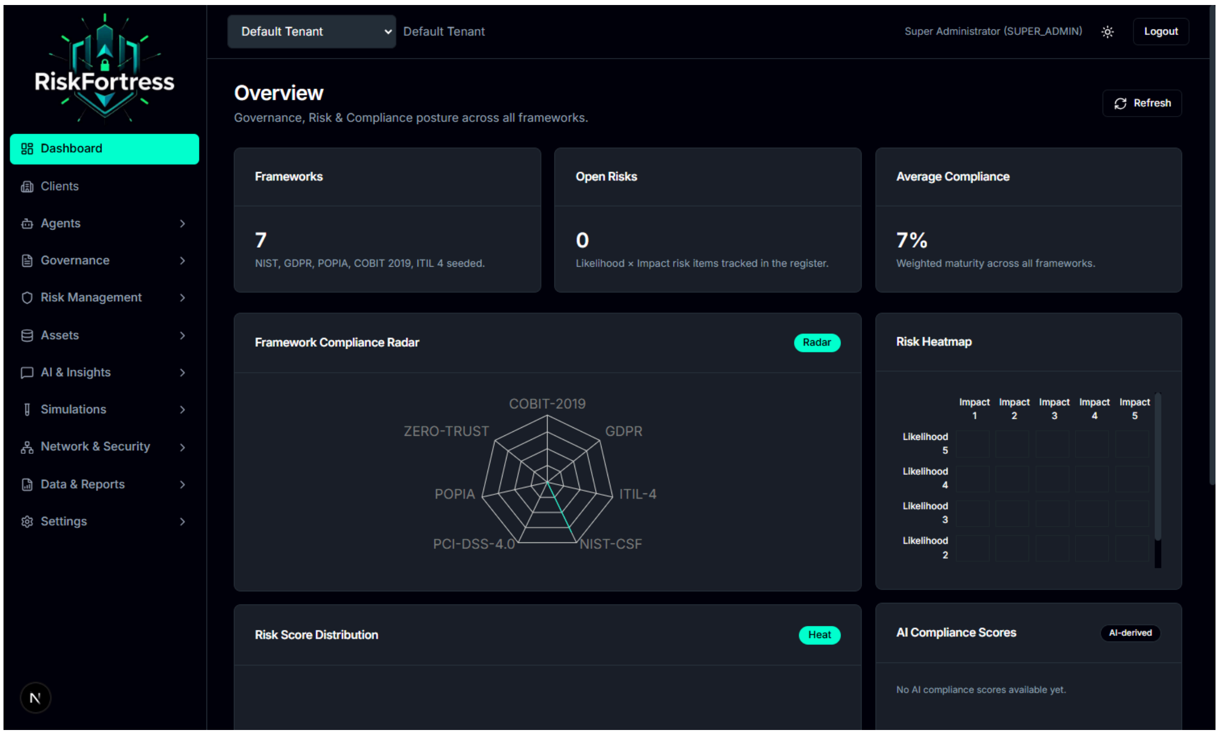
User Interface & Experience

RiskFortress provides a modern, intuitive user interface built with Next.js 15 and React 18. The application features:

- **Responsive Design**: Works seamlessly on desktop, tablet, and mobile devices
- **Dark/Light Theme**: User-selectable theme with system preference detection
- **Accessible UI**: Built on shadcn/ui components with full accessibility support
- **Real-Time Updates**: Live data updates using TanStack Query
- **Interactive Dashboards**: Rich visualizations with Recharts

Key UI Screenshots:

Figure 1: Main Dashboard - Overview of compliance posture across all frameworks



Overview

Governance, Risk & Compliance posture across all frameworks.

Frameworks

7

NIST, GDPR, POPIA, COBIT 2019, ITIL 4 seeded.

Open Risks

0

Likelihood × Impact risk items tracked in the register.

Average Compliance

7%

Weighted maturity across all frameworks.

Framework Compliance Radar

Risk Heatmap

Risk Score Distribution

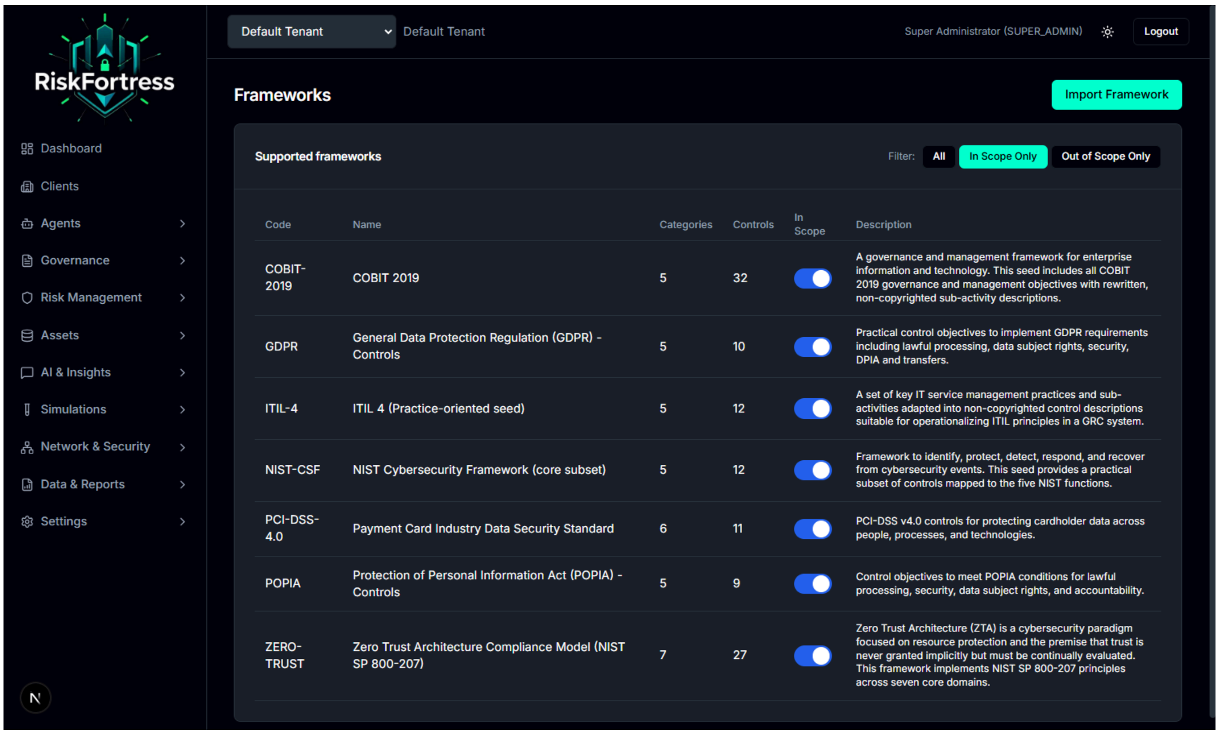
AI Compliance Scores

No AI compliance scores available yet.

The main dashboard provides a comprehensive overview of the organization's compliance posture, including:

- Framework compliance scores
- Risk register summary
- Average compliance metrics
- Framework compliance radar chart
- Risk heatmap visualization

Figure 2: Frameworks Management - Framework library and scope management



Frameworks

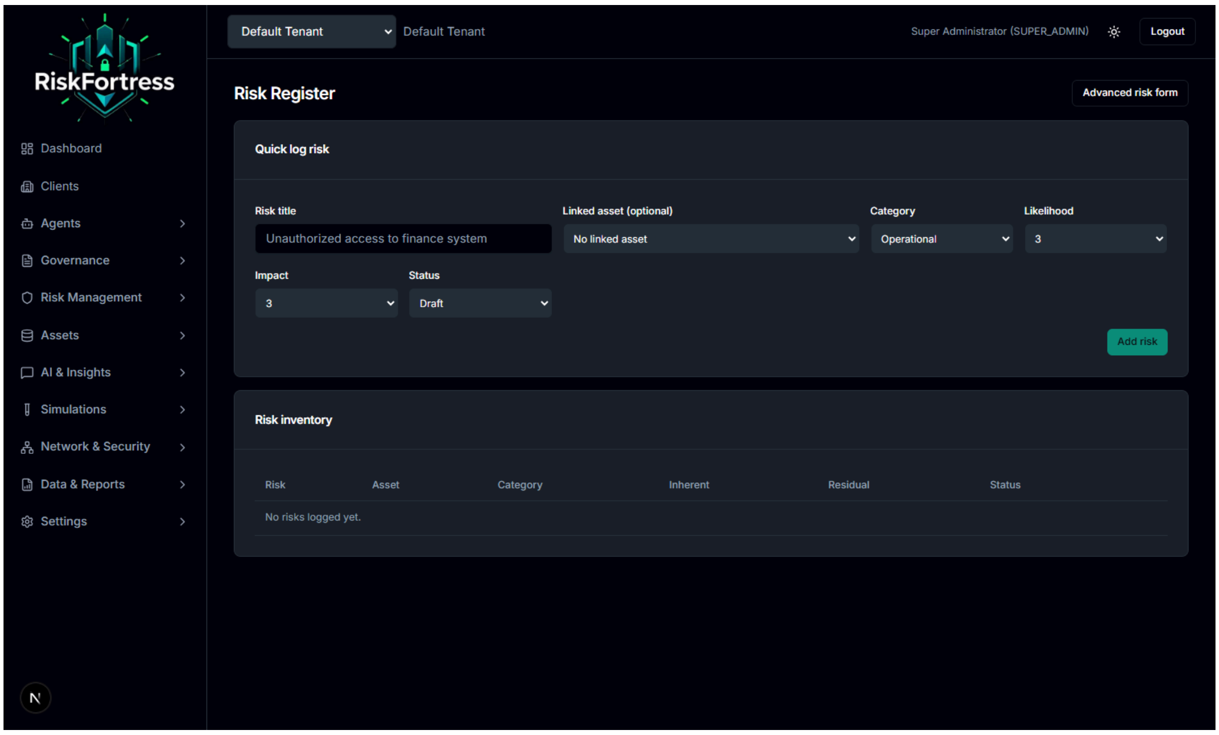
Supported frameworks

Code	Name	Categories	Controls	In Scope	Description
COBIT-2019	COBIT 2019	5	32	●	A governance and management framework for enterprise information and technology. This seed includes all COBIT 2019 governance and management objectives with rewritten, non-copyrighted sub-activity descriptions.
GDPR	General Data Protection Regulation (GDPR) - Controls	5	10	●	Practical control objectives to implement GDPR requirements including lawful processing, data subject rights, security, DPIA and transfers.
ITIL-4	ITIL 4 (Practice-oriented seed)	5	12	●	A set of key IT service management practices and sub-activities adapted into non-copyrighted control descriptions suitable for operationalizing ITIL principles in a GRC system.
NIST-CSF	NIST Cybersecurity Framework (core subset)	5	12	●	Framework to identify, protect, detect, respond, and recover from cybersecurity events. This seed provides a practical subset of controls mapped to the five NIST functions.
PCI-DSS-4.0	Payment Card Industry Data Security Standard	6	11	●	PCI-DSS v4.0 controls for protecting cardholder data across people, processes, and technologies.
POPIA	Protection of Personal Information Act (POPIA) - Controls	5	9	●	Control objectives to meet POPIA conditions for lawful processing, security, data subject rights, and accountability.
ZERO-TRUST	Zero Trust Architecture Compliance Model (NIST SP 800-207)	7	27	●	Zero Trust Architecture (ZTA) is a cybersecurity paradigm focused on resource protection and the premise that trust is never granted implicitly but must be continually evaluated. This framework implements NIST SP 800-207 principles across seven core domains.

The frameworks page displays all available compliance frameworks with:

- Framework metadata (categories, controls, descriptions)
- In-scope/out-of-scope toggle controls
- Framework import functionality
- Filtering options (All, In Scope Only, Out of Scope Only)

Figure 3: Risk Register - Risk management interface



Default Tenant

Risk Register

Quick log risk

Risk title: Unauthorized access to finance system
Linked asset (optional): No linked asset
Category: Operational
Likelihood: 3
Impact: 3
Status: Draft

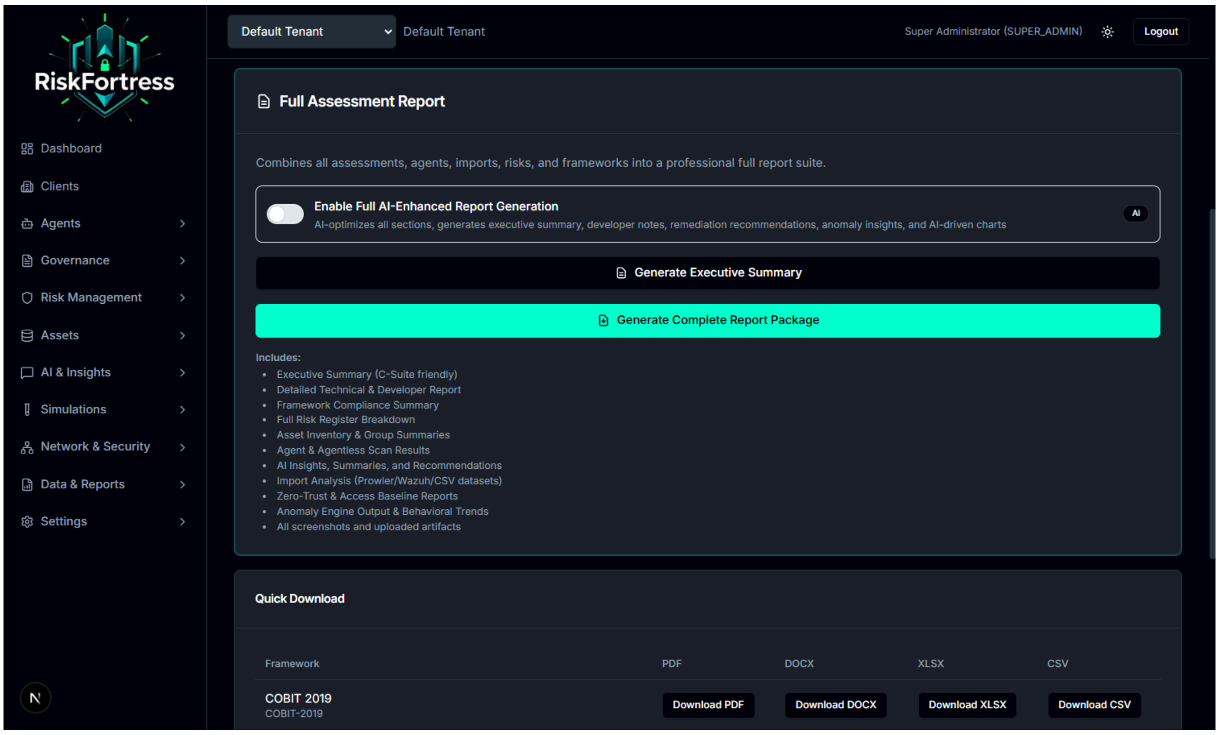
Risk inventory

Risk	Asset	Category	Inherent	Residual	Status
No risks logged yet.					

The risk register provides comprehensive risk management capabilities:

- Quick risk logging form
- Advanced risk form with detailed fields
- Risk inventory table with filtering
- Risk-to-asset linking
- Likelihood and impact scoring

Figure 4: Reports Generation - Comprehensive reporting interface



Full Assessment Report

Combines all assessments, agents, imports, risks, and frameworks into a professional full report suite.

Enable Full AI-Enhanced Report Generation
AI-optimizes all sections, generates executive summary, developer notes, remediation recommendations, anomaly insights, and AI-driven charts

Includes:
- Executive Summary (C-Suite friendly)
- Detailed Technical & Developer Report
- Framework Compliance Summary
- Full Risk Register Breakdown
- Asset Inventory & Group Summaries
- Agent & Agentless Scan Results
- AI Insights, Summaries, and Recommendations
- Import Analysis (Prowler/Wazuh/CSV datasets)
- Zero-Trust & Access Baseline Reports
- Anomaly Engine Output & Behavioral Trends
- All screenshots and uploaded artifacts

Quick Download

Framework	PDF	DOCX	XLSX	CSV
COBIT 2019 COBIT-2019	Download PDF	Download DOCX	Download XLSX	Download CSV

The reports page enables generation of professional compliance reports:

- Full assessment report package generation
- AI-enhanced report options
- Framework-specific report downloads
- Multiple export formats (PDF, DOCX, XLSX, CSV)

Next.js App Router Structure

```
app/
├── (auth)/              # Authentication route group
│   └── login/
│       └── page.tsx
├── (dashboard)/         # Main application route group
│   ├── layout.tsx       # Dashboard layout with sidebar
│   ├── dashboard/
│   │   └── page.tsx     # Main dashboard
│   ├── frameworks/
│   │   ├── page.tsx     # Framework list
│   │   └── [id]/
│   │       └── page.tsx # Framework detail
│   ├── assets/
│   │   └── page.tsx     # Asset management
│   ├── assessment/
│   │   └── [frameworkId]/
│   │       └── [controlId]/
│   │           └── page.tsx # Assessment form
│   └── ...
└── layout.tsx           # Root layout
```

Component Architecture

Component Organization:

```
components/
├── ui/                  # Base UI components (shadcn/ui)
│   ├── button.tsx
│   ├── card.tsx
│   ├── dialog.tsx
│   └── ...
├── reports/             # Report-specific components
│   ├── ReportEditor.tsx
│   ├── ReportPreview.tsx
│   └── ...
├── AIChat.tsx          # AI chat interface
├── BrandingForm.tsx    # Branding configuration
├── ComplianceScoreCard.tsx
├── RiskHeatmap.tsx
└── ...
```

State Management

Server State (TanStack Query):

```
// Example: Fetch frameworks
const { data, isLoading } = useQuery({
  queryKey: ["frameworks"],
  queryFn: async () => {
    const res = await apiRequest<Framework[]>("/frameworks");
    return res;
  }
});
```

Local State (React Hooks):

```
const [formData, setFormData] = useState({
  name: "",
  type: "server",
  // ...
});
```

URL State:

```
// Query parameters for filtering
const searchParams = useSearchParams();
const page = searchParams.get("page") || "1";
const search = searchParams.get("search") || "";
```

API Client

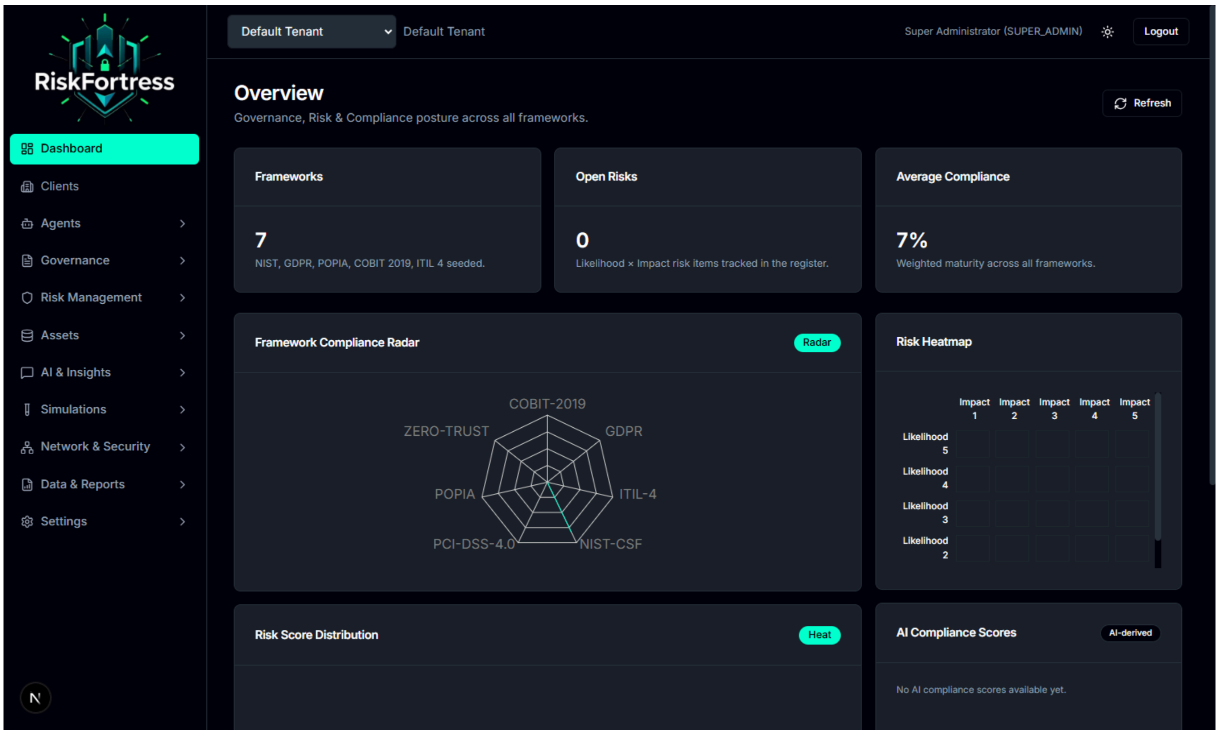
Typed API client with error handling:

```
// lib/api.ts
export async function apiRequest<T>(
  endpoint: string,
  options?: RequestInit
): Promise<T> {
  const token = localStorage.getItem("grc_token");
  const response = await fetch(`${API_URL}${endpoint}`, {
    ...options,
    headers: {
      "Content-Type": "application/json",
      ...(token && { Authorization: `Bearer ${token}` }),
      ...options?.headers,
    },
  });

  if (!response.ok) {
    throw new Error(`API error: ${response.statusText}`);
  }

  return response.json();
}
```

User Interface Screenshots

The following screenshots demonstrate the key features and capabilities of the RiskFortress platform:

Main Dashboard



Overview

Governance, Risk & Compliance posture across all frameworks.

Frameworks

7

NIST, GDPR, POPIA, COBIT 2019, ITIL 4 seeded.

Open Risks

0

Likelihood × Impact risk items tracked in the register.

Average Compliance

7%

Weighted maturity across all frameworks.

Framework Compliance Radar

Risk Heatmap

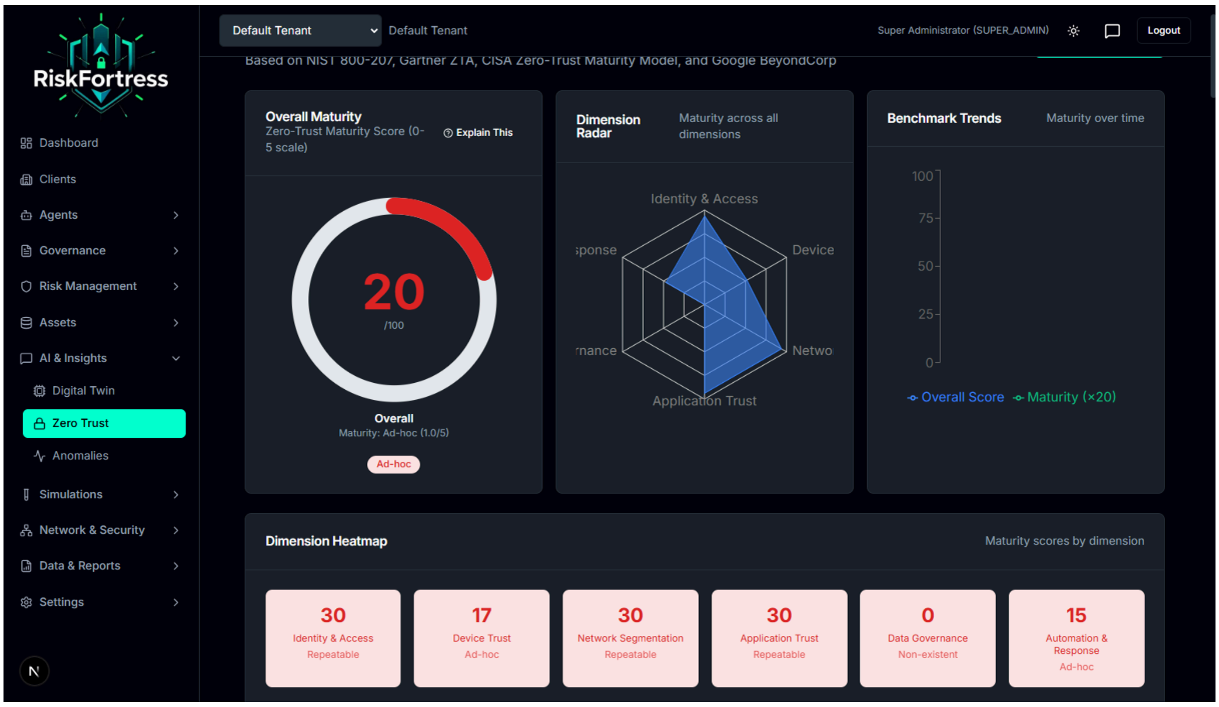
Risk Score Distribution

AI Compliance Scores

No AI compliance scores available yet.

Figure: Main dashboard showing framework compliance radar, risk score distribution, and compliance score cards. The dashboard provides a comprehensive overview of the organization's GRC posture.

Zero-Trust Maturity Dashboard



Based on NIST 800-207, Gartner ZTA, CISA Zero-Trust Maturity Model, and Google BeyondCorp

Overall Maturity
Zero-Trust Maturity Score (0-5 scale)

20
/100

Overall
Maturity: Ad-hoc (1.0/5)
Ad-hoc

Dimension Radar Maturity across all dimensions

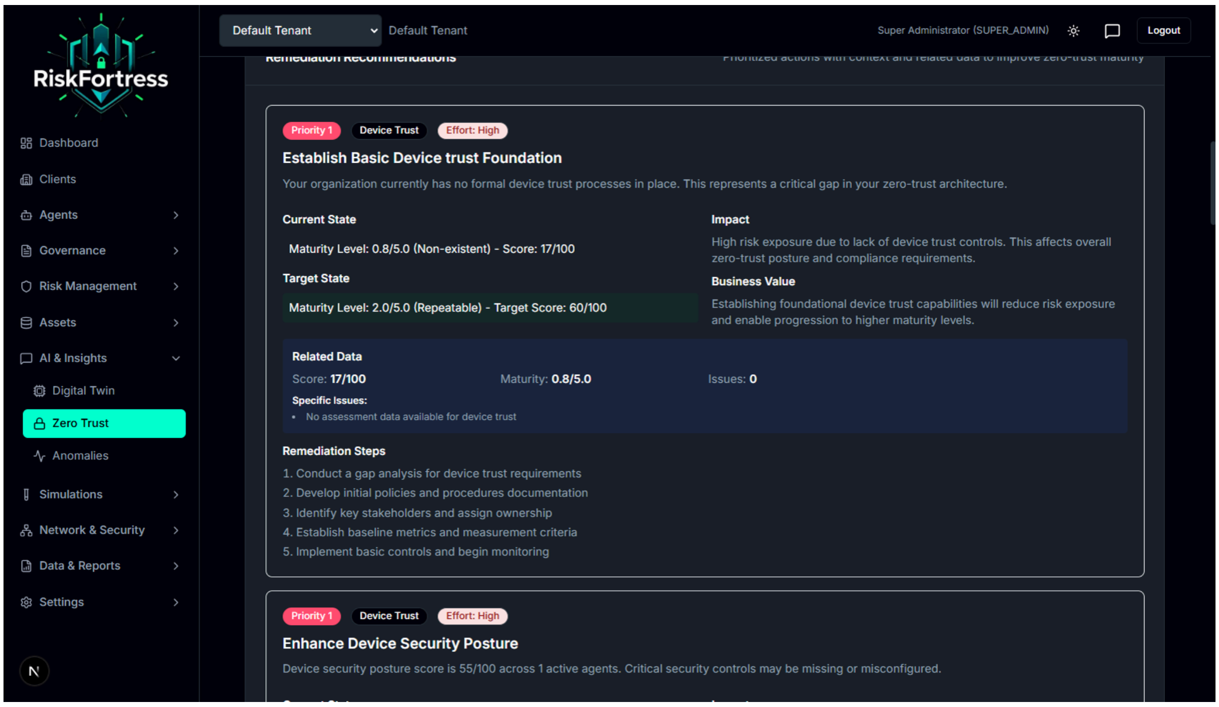
Benchmark Trends Maturity over time

Dimension Heatmap Maturity scores by dimension

30	17	30	30	0	15
Identity & Access	Device Trust	Network Segmentation	Application Trust	Data Governance	Automation & Response
Repeatable	Ad-hoc	Repeatable	Repeatable	Non-existent	Ad-hoc

Figure: Zero-Trust maturity dashboard displaying overall maturity gauge, dimension radar chart, benchmark trends, and dimension heatmap. The dashboard provides comprehensive zero-trust architecture assessment based on NIST 800-207, Gartner ZTA, CISA Zero-Trust Maturity Model, and Google BeyondCorp.

Zero-Trust Remediation Recommendations



Remediation Recommendations

Prioritized actions with context and related data to improve zero-trust maturity

Priority 1 Device Trust Effort: High

Establish Basic Device trust Foundation

Your organization currently has no formal device trust processes in place. This represents a critical gap in your zero-trust architecture.

Current State

Maturity Level: 0.8/5.0 (Non-existent) - Score: 17/100

Target State

Maturity Level: 2.0/5.0 (Repeatable) - Target Score: 60/100

Impact

High risk exposure due to lack of device trust controls. This affects overall zero-trust posture and compliance requirements.

Business Value

Establishing foundational device trust capabilities will reduce risk exposure and enable progression to higher maturity levels.

Related Data

Score: **17/100** Maturity: **0.8/5.0** Issues: **0**

Specific Issues:

- No assessment data available for device trust

Remediation Steps

1. Conduct a gap analysis for device trust requirements
2. Develop initial policies and procedures documentation
3. Identify key stakeholders and assign ownership
4. Establish baseline metrics and measurement criteria
5. Implement basic controls and begin monitoring

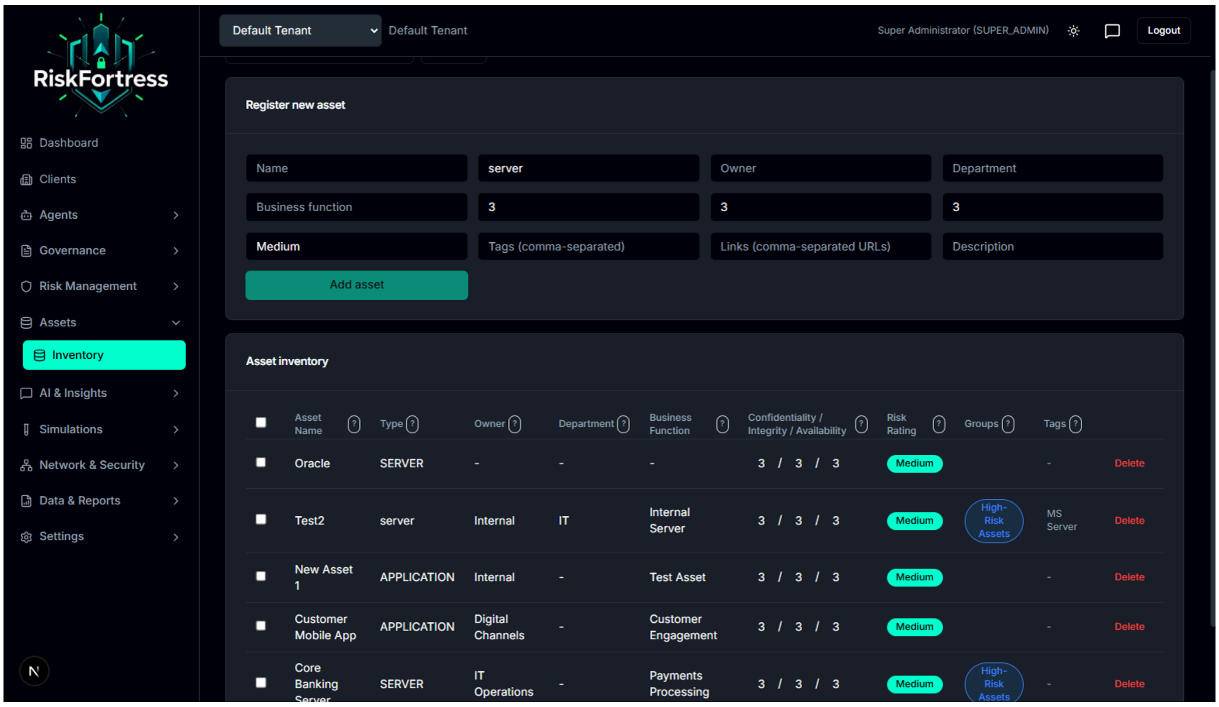
Priority 1 Device Trust Effort: High

Enhance Device Security Posture

Device security posture score is 55/100 across 1 active agents. Critical security controls may be missing or misconfigured.

Figure: Zero-Trust remediation recommendations panel showing prioritized actions with detailed context, current state, target state, impact analysis, business value, related data, and step-by-step remediation instructions. Each recommendation includes priority level, effort estimation, and specific remediation steps.

Asset Inventory Management



Register new asset

Name	server	Owner	Department
Business function	3	3	3
Medium	Tags (comma-separated)	Links (comma-separated URLs)	Description

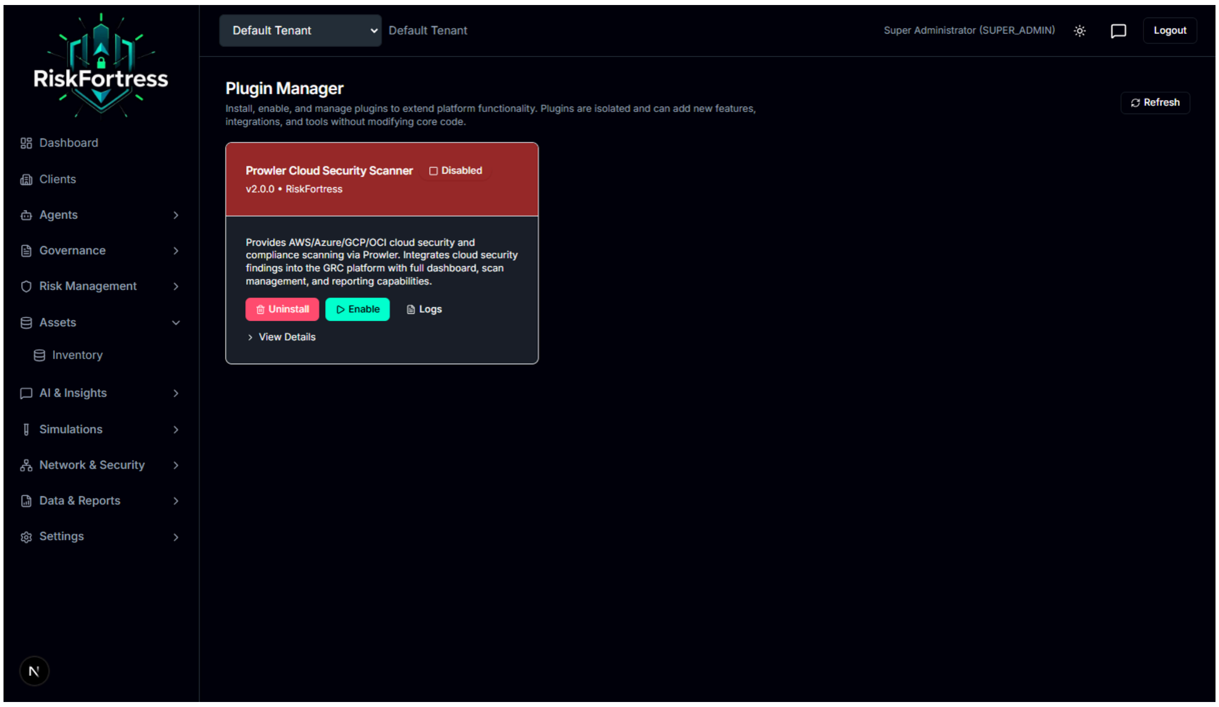
Add asset

Asset inventory

	Asset Name	Type	Owner	Department	Business Function	Confidentiality / Integrity / Availability	Risk Rating	Groups	Tags	
	Oracle	SERVER	-	-	-	3 / 3 / 3	Medium		-	Delete
	Test2	server	Internal	IT	Internal Server	3 / 3 / 3	Medium	High-Risk Assets	MS Server	Delete
	New Asset 1	APPLICATION	Internal	-	Test Asset	3 / 3 / 3	Medium		-	Delete
	Customer Mobile App	APPLICATION	Digital Channels	-	Customer Engagement	3 / 3 / 3	Medium		-	Delete
	Core Banking Server	SERVER	IT Operations	-	Payments Processing	3 / 3 / 3	Medium	High-Risk Assets	-	Delete

Figure: Asset inventory management interface showing registered assets with their types, owners, departments, business functions, confidentiality/integrity/availability ratings, risk ratings, groups, and tags. The interface includes asset registration forms and comprehensive asset inventory tables with filtering and management capabilities.

Plugin Management



Plugin Manager

Install, enable, and manage plugins to extend platform functionality. Plugins are isolated and can add new features, integrations, and tools without modifying core code.

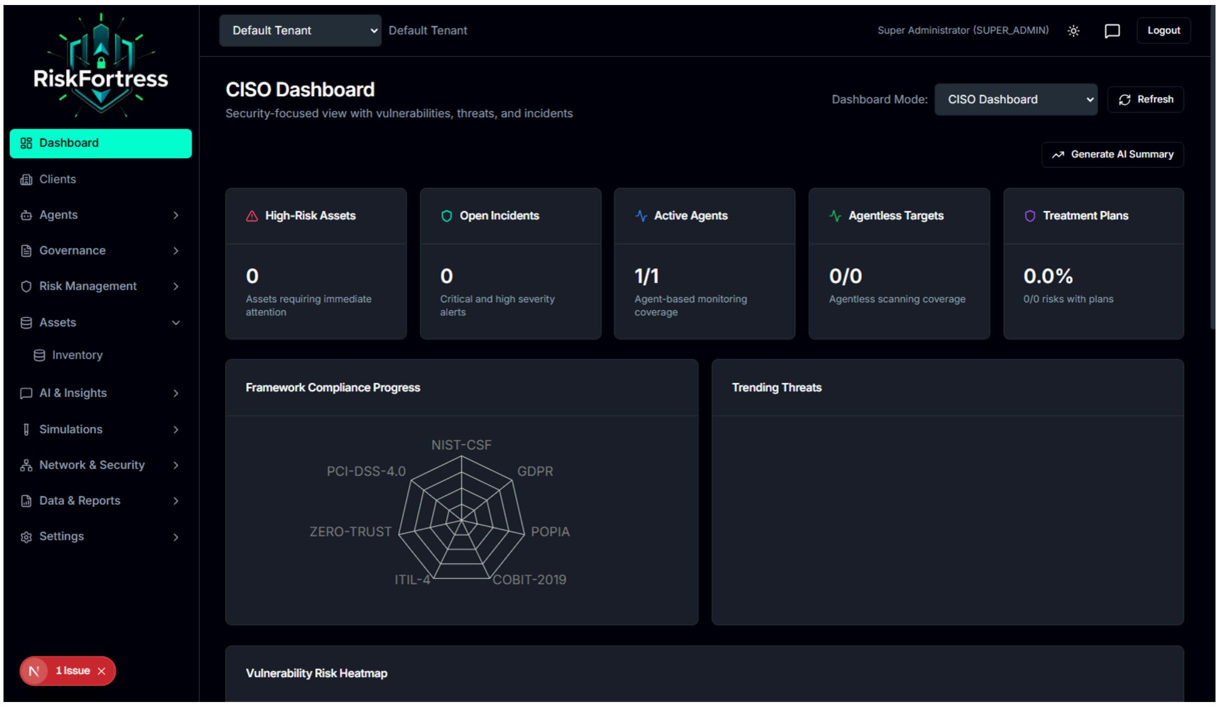
Prowler Cloud Security Scanner — Disabled

v2.0.0 • RiskFortress

Provides AWS/Azure/GCP/OCI cloud security and compliance scanning via Prowler. Integrates cloud security findings into the GRC platform with full dashboard, scan management, and reporting capabilities.

Figure: Plugin management interface displaying available plugins with version information, installation status, and management controls. The interface shows plugin details including descriptions, version numbers, vendor information, and operational status. Plugins can be installed, enabled, disabled, uninstalled, and monitored through integrated log viewing. The system supports secure plugin isolation and compatibility checking to ensure platform stability.

CISO Dashboard



CISO Dashboard

Security-focused view with vulnerabilities, threats, and incidents

High-Risk Assets
0
Assets requiring immediate attention

Open Incidents
0
Critical and high severity alerts

Active Agents
1/1
Agent-based monitoring coverage

Agentless Targets
0/0
Agentless scanning coverage

Treatment Plans
0.0%
0/0 risks with plans

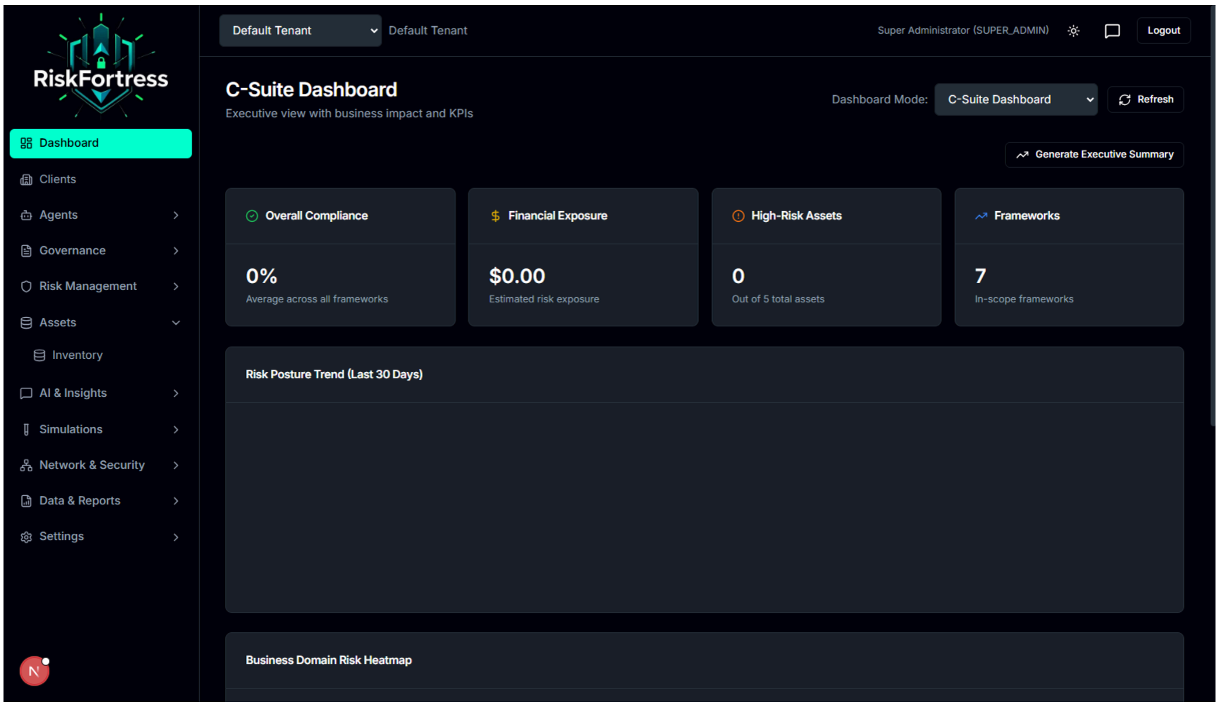
Framework Compliance Progress

Trending Threats

Vulnerability Risk Heatmap

Figure: CISO-focused dashboard providing security leaders with comprehensive visibility into high-risk assets, open incidents, active agent coverage, agentless scanning targets, and treatment plan progress. The dashboard includes framework compliance progress radar charts, trending threats analysis, and vulnerability risk heatmaps. Designed specifically for security leaders who need actionable insights into security posture, threat landscape, and compliance status across multiple frameworks.

C-Suite Dashboard



Default Tenant | Default Tenant

Super Administrator (SUPER_ADMIN)

C-Suite Dashboard

Executive view with business impact and KPIs

Dashboard Mode: C-Suite Dashboard

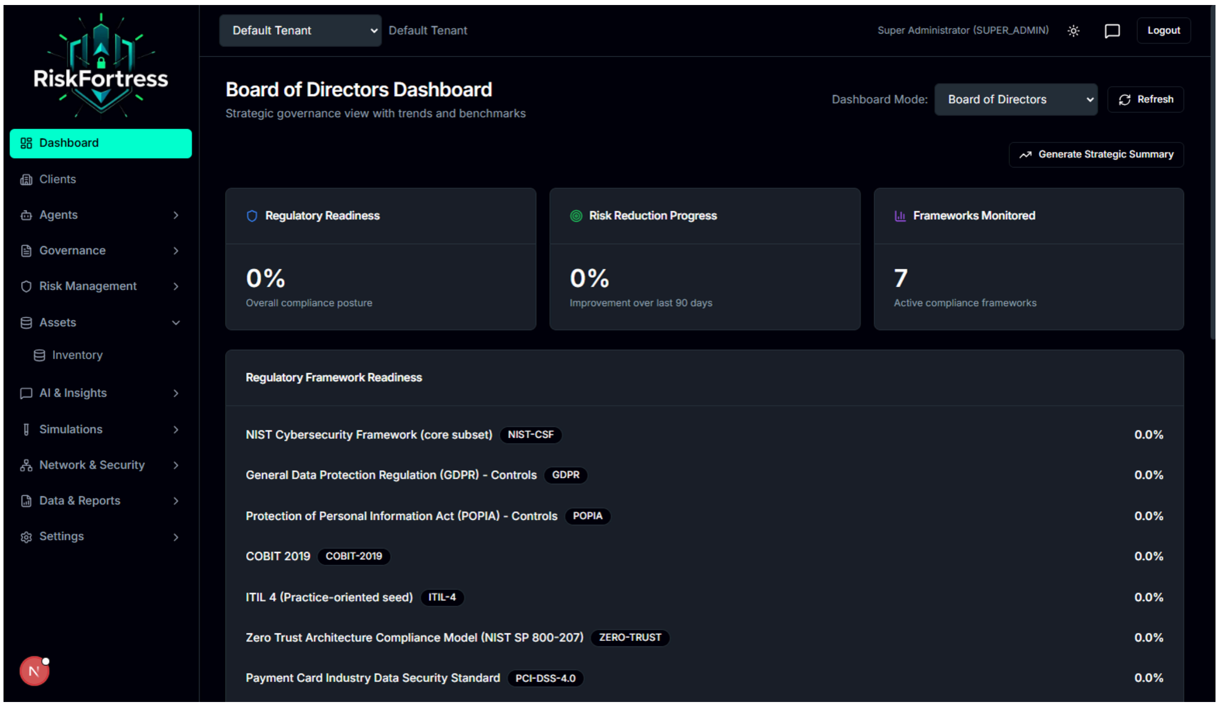
Overall Compliance	Financial Exposure	High-Risk Assets	Frameworks
0%	$0.00	0	7
Average across all frameworks	Estimated risk exposure	Out of 5 total assets	In-scope frameworks

Risk Posture Trend (Last 30 Days)

Business Domain Risk Heatmap

Figure: Executive dashboard emphasizing business impact with key metrics including overall compliance percentage, financial exposure estimates, high-risk asset counts, and framework coverage. The dashboard includes risk posture trend analysis over the last 30 days and business domain risk heatmaps. Optimized for C-level executives who need business-focused insights without technical complexity, enabling strategic decision-making based on compliance posture and risk exposure.

Board of Directors Dashboard



Default Tenant

Board of Directors Dashboard

Strategic governance view with trends and benchmarks

Dashboard Mode: Board of Directors

Regulatory Readiness
0%
Overall compliance posture

Risk Reduction Progress
0%
Improvement over last 90 days

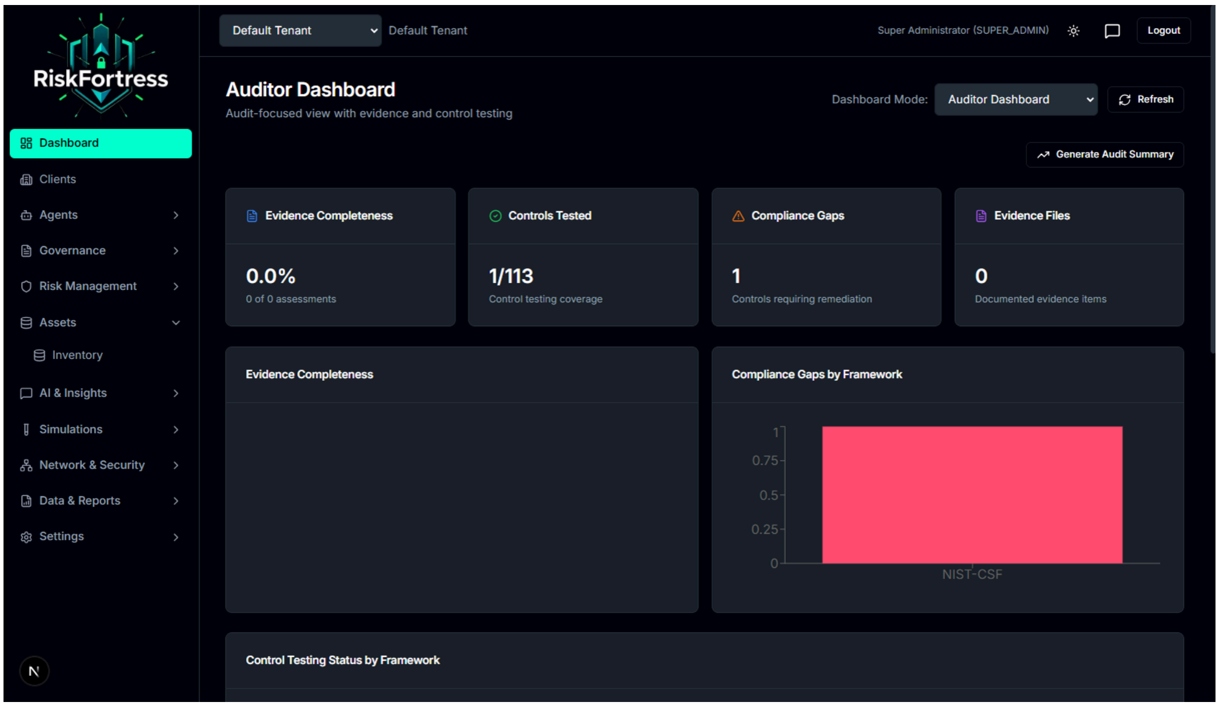
Frameworks Monitored
7
Active compliance frameworks

Regulatory Framework Readiness

NIST Cybersecurity Framework (core subset) NIST-CSF	0.0%
General Data Protection Regulation (GDPR) - Controls GDPR	0.0%
Protection of Personal Information Act (POPIA) - Controls POPIA	0.0%
COBIT 2019 COBIT-2019	0.0%
ITIL 4 (Practice-oriented seed) ITIL-4	0.0%
Zero Trust Architecture Compliance Model (NIST SP 800-207) ZERO-TRUST	0.0%
Payment Card Industry Data Security Standard PCI-DSS-4.0	0.0%

Figure: Board-level dashboard providing strategic governance insights with regulatory readiness percentages, risk reduction progress metrics, and comprehensive framework monitoring. The dashboard displays regulatory framework readiness across all active frameworks (NIST-CSF, GDPR, POPIA, COBIT-2019, ITIL-4, ZERO-TRUST, PCI-DSS-4.0) with individual readiness scores. Designed for governance and strategic oversight, enabling board members to assess organizational compliance posture and regulatory alignment at a glance.

Auditor Dashboard



Auditor Dashboard

Audit-focused view with evidence and control testing

Dashboard Mode: Auditor Dashboard

Evidence Completeness
0.0%
0 of 0 assessments

Controls Tested
1/113
Control testing coverage

Compliance Gaps
1
Controls requiring remediation

Evidence Files
0
Documented evidence items

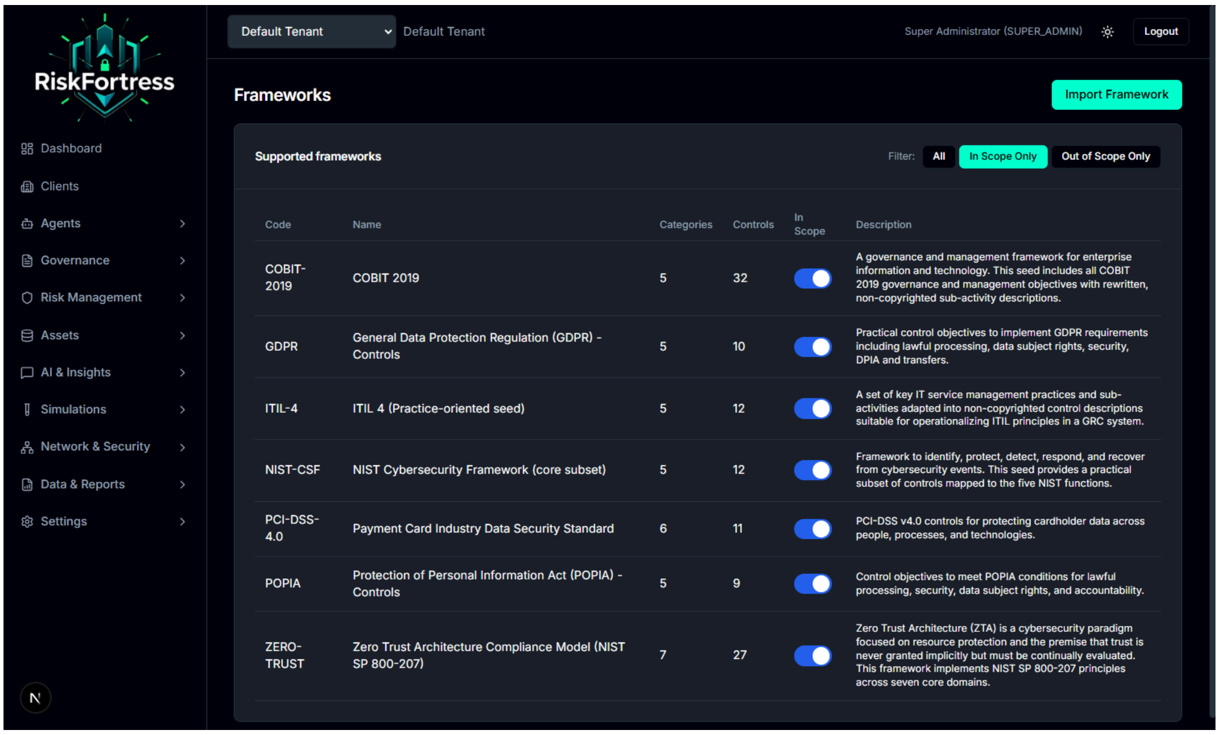
Evidence Completeness

Compliance Gaps by Framework

Control Testing Status by Framework

Figure: Auditor-focused dashboard displaying evidence completeness percentages, control testing coverage (controls tested vs. total controls), compliance gaps count, and evidence file counts. The dashboard includes evidence completeness panels, compliance gaps by framework bar charts, and control testing status by framework analysis. Optimized for audit preparation and evidence collection, enabling auditors to quickly assess audit readiness, identify gaps, and track evidence documentation progress across all compliance frameworks.

Frameworks Page



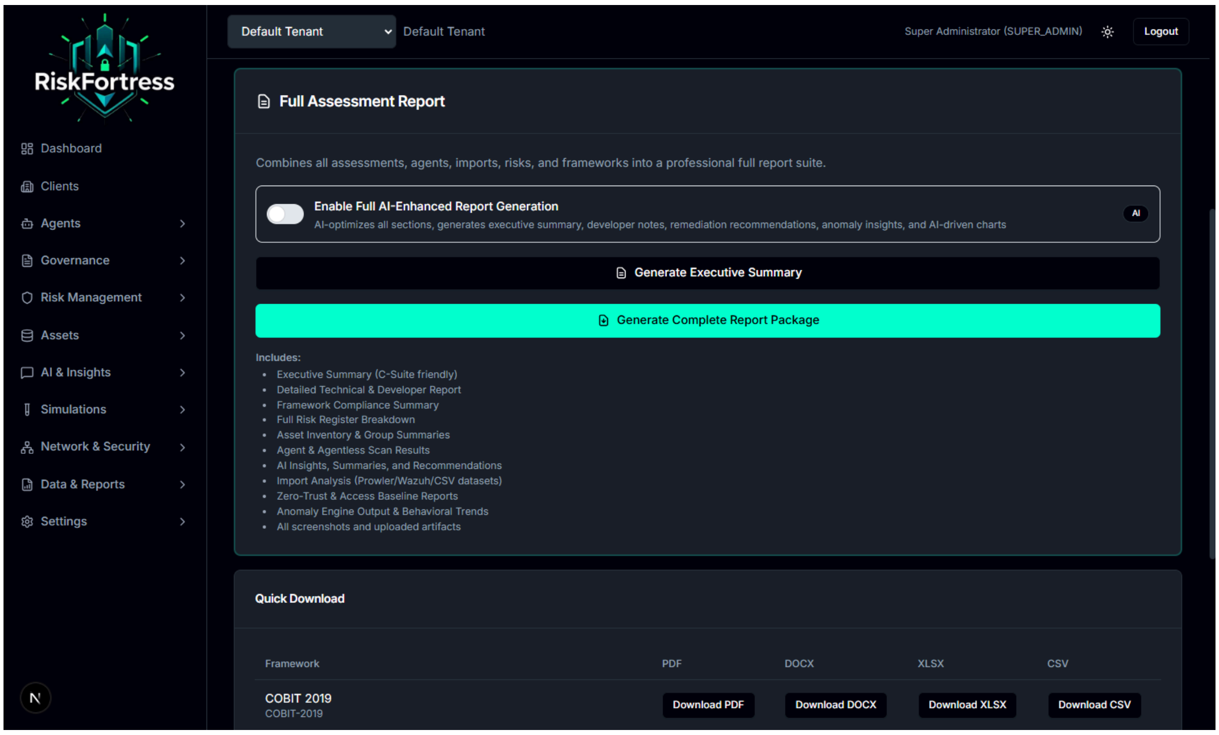
Frameworks

Supported frameworks

Code	Name	Categories	Controls	In Scope	Description
COBIT-2019	COBIT 2019	5	32	●	A governance and management framework for enterprise information and technology. This seed includes all COBIT 2019 governance and management objectives with rewritten, non-copyrighted sub-activity descriptions.
GDPR	General Data Protection Regulation (GDPR) - Controls	5	10	●	Practical control objectives to implement GDPR requirements including lawful processing, data subject rights, security, DPIA and transfers.
ITIL-4	ITIL 4 (Practice-oriented seed)	5	12	●	A set of key IT service management practices and sub-activities adapted into non-copyrighted control descriptions suitable for operationalizing ITIL principles in a GRC system.
NIST-CSF	NIST Cybersecurity Framework (core subset)	5	12	●	Framework to identify, protect, detect, respond, and recover from cybersecurity events. This seed provides a practical subset of controls mapped to the five NIST functions.
PCI-DSS-4.0	Payment Card Industry Data Security Standard	6	11	●	PCI-DSS v4.0 controls for protecting cardholder data across people, processes, and technologies.
POPIA	Protection of Personal Information Act (POPIA) - Controls	5	9	●	Control objectives to meet POPIA conditions for lawful processing, security, data subject rights, and accountability.
ZERO-TRUST	Zero Trust Architecture Compliance Model (NIST SP 800-207)	7	27	●	Zero Trust Architecture (ZTA) is a cybersecurity paradigm focused on resource protection and the premise that trust is never granted implicitly but must be continually evaluated. This framework implements NIST SP 800-207 principles across seven core domains.

Figure: Framework management interface showing all available compliance frameworks (NIST, GDPR, POPIA, COBIT, ITIL) with scope management, category navigation, and control details.

Reports Page



Full Assessment Report

Combines all assessments, agents, imports, risks, and frameworks into a professional full report suite.

Enable Full AI-Enhanced Report Generation
AI-optimizes all sections, generates executive summary, developer notes, remediation recommendations, anomaly insights, and AI-driven charts
AI

Generate Executive Summary

Generate Complete Report Package

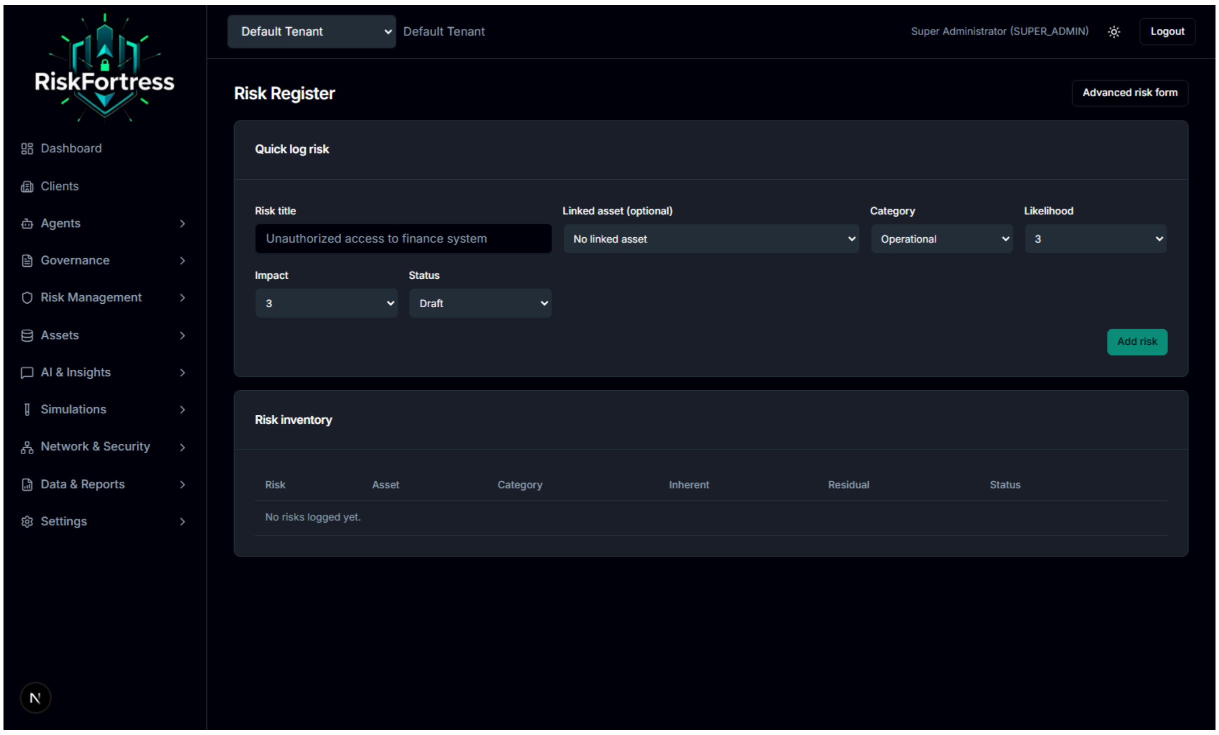
Includes:
- Executive Summary (C-Suite friendly)
- Detailed Technical & Developer Report
- Framework Compliance Summary
- Full Risk Register Breakdown
- Asset Inventory & Group Summaries
- Agent & Agentless Scan Results
- AI Insights, Summaries, and Recommendations
- Import Analysis (Prowler/Wazuh/CSV datasets)
- Zero-Trust & Access Baseline Reports
- Anomaly Engine Output & Behavioral Trends
- All screenshots and uploaded artifacts

Quick Download

Framework	PDF	DOCX	XLSX	CSV
COBIT 2019 COBIT-2019	Download PDF	Download DOCX	Download XLSX	Download CSV

Figure: Report generation interface with options for framework-specific reports, executive summaries, technical reports, and audit packages. Supports multiple export formats including PDF, HTML, Word, and Excel.

Risk Register



Risk Register

Quick log risk

Risk title	Unauthorized access to finance system
Linked asset (optional)	No linked asset
Category	Operational
Likelihood	3
Impact	3
Status	Draft

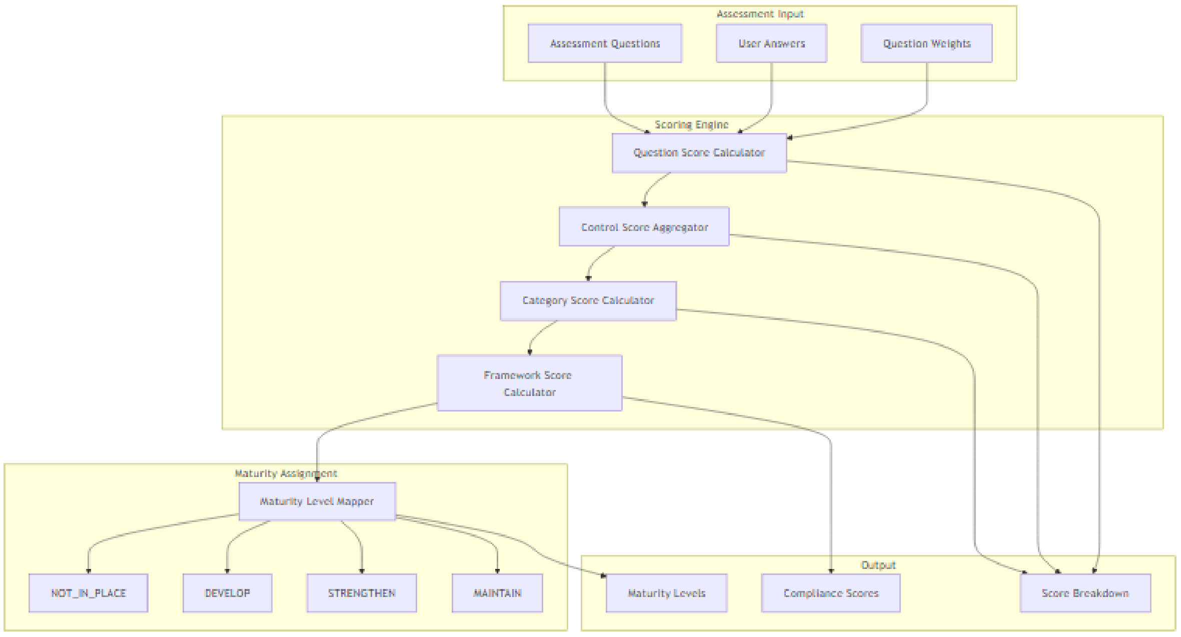
Risk inventory

Risk	Asset	Category	Inherent	Residual	Status
No risks logged yet.					

Figure: Risk register interface displaying risk items with likelihood-impact scoring, risk heatmap visualization, mitigation tracking, and risk-to-control mapping.

Scoring Engine Implementation

Compliance Scoring Flow



Scoring Algorithm

The scoring engine calculates compliance scores at multiple levels:

Question-Level Scoring:

```
function calculateQuestionScore(
  question: Question,
  answer: Answer
): number {
  switch (question.type) {
    case "boolean":
      return answer === true ? question.weight : 0;
    case "multiple-choice":
      const option = question.options.find(o => o.value === answer);
      return option?.weight || 0;
    case "text":
      return 0; // Manual scoring required
  }
}
```

Control-Level Aggregation:

```
function calculateControlScore(
  questions: Question[],
  answers: Answer[]
): number {
  const totalWeight = questions.reduce((sum, q) => sum + q.weight, 0);
  const scoredWeight = questions.reduce((sum, q) => {
    const answer = answers.find(a => a.questionId === q.id);
    return sum + calculateQuestionScore(q, answer) * q.weight;
  }, 0);

  return totalWeight > 0 ? scoredWeight / totalWeight : 0;
}
```

Framework-Level Aggregation:

```
function calculateFrameworkScore(
  categories: Category[]
): number {
  const categoryScores = categories.map(cat =>
    calculateCategoryScore(cat)
  );
  return categoryScores.reduce((sum, score) => sum + score, 0)
    / categoryScores.length;
}
```

Maturity Level Assignment

```
function assignMaturityLevel(score: number): MaturityLevel {
  if (score >= 0.76) return MaturityLevel.MAINTAIN;
  if (score >= 0.51) return MaturityLevel.STRENGTHEN;
  if (score >= 0.26) return MaturityLevel.DEVELOP;
  return MaturityLevel.NOT_IN_PLACE;
}
```

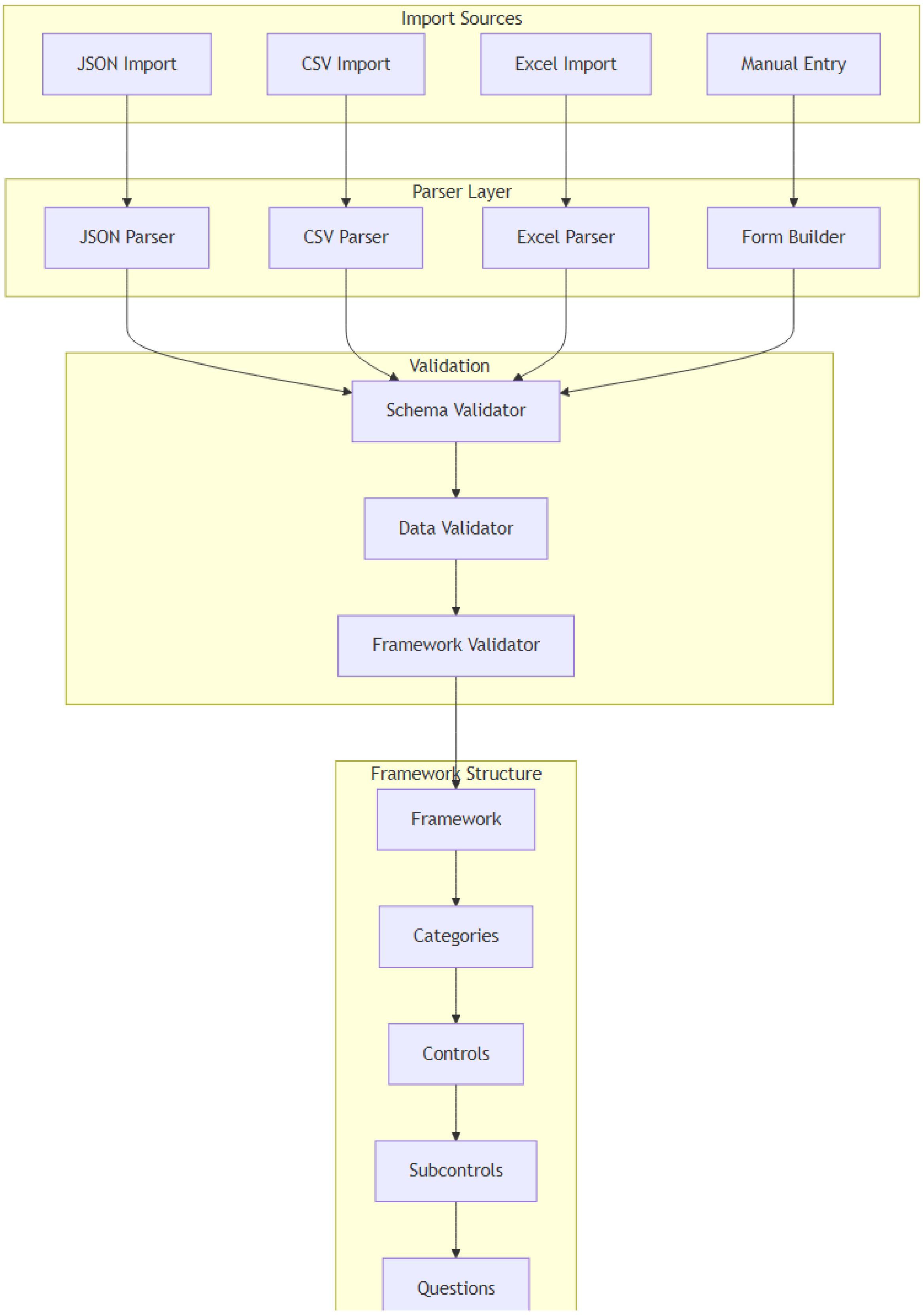
Conditional Logic

Questions can be conditionally displayed based on previous answers:

```
function shouldShowQuestion(
  question: Question,
  previousAnswers: Answer[]
): boolean {
  if (!question.condition) return true;

  const dependsOnAnswer = previousAnswers.find(
    a => a.questionId === question.condition.dependsOn
  );

  return dependsOnAnswer?.value === question.condition.expectedAnswer;
}
```

Integration Architecture

Framework Import & Mapping Pipeline


Import Sources
JSON Import
CSV Import
Excel Import
Manual Entry
Parser Layer
JSON Parser
CSV Parser
Excel Parser
Form Builder
Validation
Schema Validator
Data Validator
Framework Validator
Framework Structure
Framework
Categories
Controls
Subcontrols
Questions

Prowler Integration

Parser Implementation:

```
export function parseProwlerJSON(content: string): ParseResult {
  const data = JSON.parse(content);
  const checks = Array.isArray(data) ? data : data.checks || [];

  const findings: ParsedFinding[] = checks.map(check => ({
    checkId: check.check_id,
    title: check.check_title,
    status: normalizeStatus(check.status),
    severity: normalizeSeverity(check.severity),
    framework: check.framework || "unknown",
    // ...
  }));

  return {
    source: "prowler",
    findings,
    frameworks: extractFrameworks(findings)
  };
}
```

Wazuh Integration

Parser Implementation:

```
export function parseWazuhJSON(content: string): ParseResult {
  const data = JSON.parse(content);
  const findings: ParsedFinding[] = data.map(item => ({
    checkId: item.rule_id,
    title: item.description,
    status: item.compliance === "PASS" ? "PASS" : "FAIL",
    framework: detectFramework(item),
    // ...
  }));

  return {
    source: "wazuh",
    findings,
    frameworks: extractFrameworks(findings)
  };
}
```

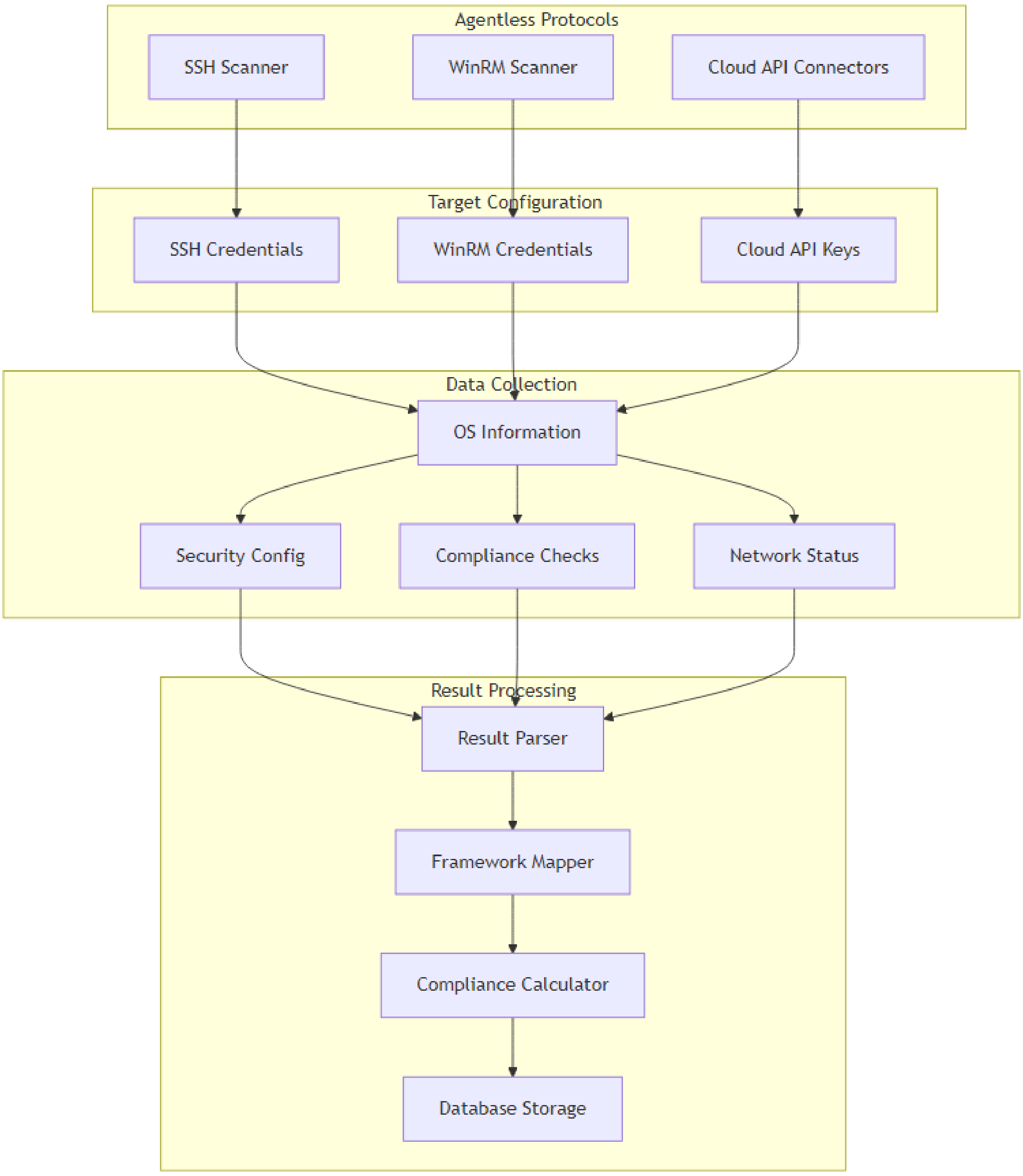
Auto-Answering Logic

```
async function autoAnswerAssessments(
  findings: ParsedFinding[],
  prisma: PrismaClient
): Promise<void> {
  for (const finding of findings) {
    // Find matching control
    const control = await prisma.frameworkControl.findFirst({
      where: {
        controlId: { contains: finding.checkId }
      }
    });

    if (control) {
      // Find or create assessment
      const assessment = await prisma.assessment.upsert({
        where: { /* ... */ },
        update: {
          maturity: finding.status === "PASS" ? 4 : 1,
          evidence: finding.metadata
        },
        create: { /* ... */ }
      });

      // Auto-answer questions
      await autoAnswerQuestions(assessment, finding, prisma);
    }
  }
}
```

Agentless Scanning Architecture

RiskFortress supports agentless scanning for systems where installing agents is not feasible:

Agentless Scanning Flow:



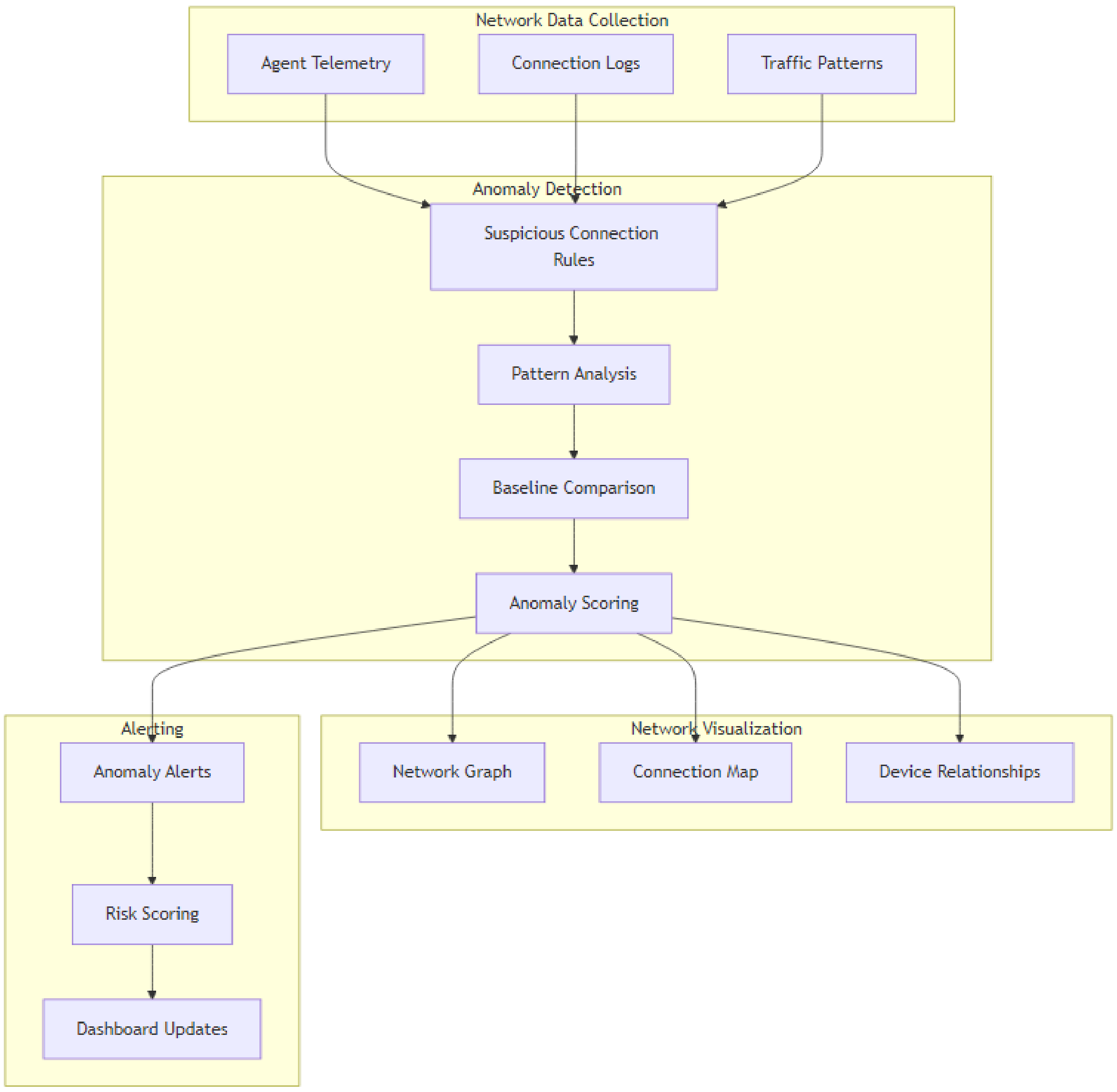
Supported Protocols:

- **SSH (Linux/Mac)**: Private key or password authentication, collects OS info, packages, services, security configs
- **WinRM (Windows)**: Basic authentication with HTTPS, collects Windows updates, security settings, compliance status
- **Cloud APIs**: AWS, Azure, GCP API connectors for cloud resource compliance scanning

Network Anomaly Detection & Visualization

RiskFortress includes advanced network monitoring and anomaly detection:

Network Anomaly Detection:



Key Features:

- **Suspicious Connection Detection**: Identifies unusual network connections based on configurable rules
- **Network Graph Visualization**: Interactive device-to-device communication mapping

- **Baseline Comparison**: Compares current network patterns against established baselines
- **Real-Time Monitoring**: Continuous monitoring with configurable alert thresholds

Network Compliance Scoring

The platform calculates network security compliance scores based on connection patterns and security configurations:

Network Compliance Calculation:

- Analyzes device-to-device communication patterns
- Evaluates connection security (encrypted vs. unencrypted)
- Assesses network segmentation compliance
- Calculates zero-trust network maturity
- Integrates with framework compliance scoring

Security Architecture

Authentication

JWT Token Generation:

```
app.post("/auth/login", async (request, reply) => {
  const { email, password } = request.body;

  const user = await prisma.user.findUnique({ where: { email } });
  if (!user) return reply.code(401).send({ error: "Invalid credentials" })

  const valid = await bcrypt.compare(password, user.password);
  if (!valid) return reply.code(401).send({ error: "Invalid credentials" }

  const token = app.jwt.sign({
    id: user.id,
    email: user.email,
    role: user.role
  });

  return { token, user: { id: user.id, email: user.email, role: user.role
});
```

Route Protection:

```
app.addHook("onRequest", async (request, reply) => {
  try {
    await request.jwtVerify();
  } catch {
    reply.code(401).send({ error: "Unauthorized" });
  }
});
```

Authorization

Role-Based Access Control:

```
function requireRole(roles: UserRole[]) {
  return async (request: FastifyRequest, reply: FastifyReply) => {
    const user = request.user;
    if (!roles.includes(user.role)) {
      return reply.code(403).send({ error: "Forbidden" });
    }
  };
}

// Usage
app.post("/frameworks", {
  preHandler: requireRole([UserRole.ADMIN, UserRole.ANALYST])
}, async (request, reply) => {
  // ...
});
```

Data Encryption

Credential Encryption:

```
import crypto from "crypto";

function encrypt(text: string, key: string): string {
  const iv = crypto.randomBytes(16);
  const cipher = crypto.createCipheriv("aes-256-cbc", Buffer.from(key, "he
  let encrypted = cipher.update(text);
  encrypted = Buffer.concat([encrypted, cipher.final()]);
  return iv.toString("hex") + ":" + encrypted.toString("hex");
}

function decrypt(encrypted: string, key: string): string {
  const parts = encrypted.split(":");
  const iv = Buffer.from(parts[0], "hex");
  const encryptedText = Buffer.from(parts[1], "hex");
  const decipher = crypto.createDecipheriv("aes-256-cbc", Buffer.from(key,
  let decrypted = decipher.update(encryptedText);
  decrypted = Buffer.concat([decrypted, decipher.final()]);
  return decrypted.toString();
}
```

Multi-Tenancy Architecture

Tenant Isolation:

```
// Tenant model with encryption key
model Tenant {
  id          String   @id @default(cuid())
  name        String
  description String?
  status      String   @default("active")
  users       User[]
  encryptionKey TenantEncryptionKey?
}

model TenantEncryptionKey {
  id            String   @id @default(cuid())
  tenantId      String   @unique
  encryptedKey  String   // Master key encrypted with system key
  keyVersion    Int      @default(1)
}
```

Tenant Middleware:

```
export async function tenantIsolationMiddleware(
  request: FastifyRequest,
  reply: FastifyReply,
  prisma: PrismaClient
): Promise<void> {
  const user = (request as any).user;
  const tenantId = extractTenantId(request, user);

  // Verify tenant exists and is active
  const tenant = await prisma.tenant.findUnique({
    where: { id: tenantId }
  });

  // Check access permissions
  if (!canAccessTenant(user.role, user.tenantId, tenantId)) {
    return reply.code(403).send({ error: "Forbidden" });
  }

  // Attach tenant context
  (request as any).tenantContext = { tenantId, user };
}
```

Per-Tenant Encryption:

```
export async function encryptForTenant(
  data: string,
  tenantId: string
): Promise<string> {
  const tenantKey = await getTenantEncryptionKey(tenantId);
  return encrypt(data, tenantKey);
}
```

RBAC Permissions System:

```
// Permission definitions
const PERMISSIONS = {
  FRAMEWORKS_READ: ["ADMIN", "ANALYST", "AUDITOR", "VIEWER"],
  FRAMEWORKS_WRITE: ["ADMIN", "ANALYST"],
  RISKS_READ: ["ADMIN", "ANALYST", "AUDITOR", "VIEWER"],
  RISKS_WRITE: ["ADMIN", "ANALYST"],
  REPORTS_GENERATE: ["ADMIN", "ANALYST", "AUDITOR"]
};

export function requirePermission(permission: string) {
  return async (request: FastifyRequest, reply: FastifyReply) => {
    const user = (request as any).user;
    const allowedRoles = PERMISSIONS[permission] || [];
    if (!allowedRoles.includes(user.role)) {
      return reply.code(403).send({ error: "Forbidden" });
    }
  };
}
```

Advanced Feature Architecture

Timeline & Roadmap Service

Gantt Timeline Implementation:

```
// Control timeline model
model ControlTimeline {
  id                   String   @id @default(cuid())
  tenantId             String
  controlId            String
  frameworkId          String
  startDate            DateTime
  expectedCompletion   DateTime
  actualCompletion     DateTime?
  status               String   // "not_started" | "in_progress" | "complet
  dependencies         Json     // Array of control IDs
  priority             String
  estimatedEffort     Int?
}

// Roadmap service
export async function getControlTimeline(
  prisma: PrismaClient,
  tenantId: string,
  frameworkId?: string
): Promise<ControlTimelineItem[]> {
  const timeline = await prisma.controlTimeline.findMany({
    where: { tenantId, frameworkId },
    include: { control: true, framework: true }
  });

  // Update status based on dates
  return timeline.map(item => ({
    ...item,
    status: calculateStatus(item)
  }));
}
```

Compliance Forecasting Engine:

```
export async function predictComplianceScore(
  prisma: PrismaClient,
  daysAhead: number
): Promise<PredictedScores> {
  const currentState = await calculateCurrentComplianceState(prisma);
  const controlAging = await analyzeControlAging(prisma);

  // Calculate degradation factors
  const agingImpact = controlAging.summary.criticalAging * 0.5 +
                      controlAging.summary.highAging * 0.3;

  // Predict scores
  const predictedFrameworkScores: Record<string, number> = {};
  for (const [frameworkId, currentScore] of Object.entries(currentState.fr
    const agingFactor = (agingImpact / 100) * (daysAhead / 30);
    predictedFrameworkScores[frameworkId] = Math.max(0, currentScore - agi
  }

  return {
    overallCompliance: Object.values(predictedFrameworkScores).reduce((a,
                       Object.keys(predictedFrameworkScores).length,
    frameworkScores: predictedFrameworkScores,
    confidence: calculateConfidence(controlAging)
  };
}
```

Board-Level Dashboard Architecture

Dashboard Mode System:

```
type DashboardMode = "CISO" | "C-SUITE" | "BOARD" | "AUDITOR" | "DEFAULT";

// Dashboard data aggregation
export async function getDashboardData(
  prisma: PrismaClient,
  mode: DashboardMode
): Promise<DashboardData> {
  switch (mode) {
    case "CISO":
      return {
        vulnerabilities: await getVulnerabilities(prisma),
        threats: await getThreats(prisma),
        incidents: await getIncidents(prisma),
        agentCoverage: await getAgentCoverage(prisma),
        frameworkScores: await getFrameworkScores(prisma)
      };

    case "C-SUITE":
      return {
        businessImpact: await calculateBusinessImpact(prisma),
        financialExposure: await calculateFinancialExposure(prisma),
        kpis: await getKPIs(prisma),
        strategicRecommendations: await getStrategicRecommendations(prisma
      };

    case "BOARD":
      return {
        regulatoryReadiness: await getRegulatoryReadiness(prisma),
        riskReductionProgress: await getRiskReductionProgress(prisma),
        yearOverYearTrends: await getYearOverYearTrends(prisma),
        departmentalBenchmarks: await getDepartmentalBenchmarks(prisma)
      };

    case "AUDITOR":
      return {
        evidenceCompleteness: await getEvidenceCompleteness(prisma),
        controlTestingCoverage: await getControlTestingCoverage(prisma),
        complianceGaps: await getComplianceGaps(prisma),
        auditReadiness: await calculateAuditReadiness(prisma)
      };
    }
  }
```

AI Dashboard Summary Generation:

```
export async function generateDashboardSummary(
  mode: DashboardMode,
  dashboardData: any
): Promise<string> {
  const prompts = {
    CISO: `Generate a CISO dashboard summary focusing on critical vulnerab
           security posture, threat landscape, and agent coverage.`,
    "C-SUITE": `Generate an executive summary focusing on business impact,
                financial exposure, and strategic recommendations.`,
    BOARD: `Generate a board-level summary focusing on regulatory readines
            risk reduction progress, and strategic governance.`,
    AUDITOR: `Generate an auditor summary focusing on evidence completenes
              control testing coverage, and compliance gaps.`
  };

  return await callAI(prompts[mode], { dashboardData });
}
```

Risk Treatment Plan Service

Treatment Plan Generation:

```
export async function generateRiskTreatmentPlan(
  prisma: PrismaClient,
  riskId: string
): Promise<RiskTreatmentPlan> {
  // 1. Evaluate risk
  const evaluation = await evaluateRiskForTreatment(prisma, riskId);

  // 2. Generate recommendation
  const recommendation = await generateTreatmentRecommendation(risk, evalu

  // 3. Generate plan content
  const planContent = await generatePlanContent(risk, evaluation, recommen

  // 4. Save to database
  return await prisma.riskTreatmentPlan.create({
    data: {
      riskId,
      recommendation: recommendation.recommendation,
      recommendationDetails: recommendation as any,
      costAnalysis: evaluation.costAnalysis,
      impactAnalysis: evaluation.impactAnalysis,
      planContent
    }
  });
}

// Cost-impact modeling
function calculateCostImpact(
  controls: Control[],
  risk: RiskItem
): CostImpactAnalysis {
  const costPerControl = controls.map(control => ({
    controlId: control.id,
    estimatedCost: estimateControlCost(control),
    costBreakdown: getCostBreakdown(control)
  }));

  const totalCost = costPerControl.reduce((sum, c) => sum + c.estimatedCos
  return {
    estimatedCost: totalCost,
    costPerControl,
    totalCostRange: {
      min: totalCost * 0.8,
      max: totalCost * 1.5
    }
  };
}
```

MSP Portfolio Dashboard

Portfolio Service Architecture:

```
// Main database for client management
model Client {
  id           String   @id @default(cuid())
  name         String
  databasePath String   @unique
  status       String   @default("active")
}

// Portfolio overview service
export async function getPortfolioOverview(
  mainPrisma: PrismaClient,
  sortBy?: "risk" | "compliance" | "maturity" | "name"
): Promise<PortfolioOverview> {
  const clients = await mainPrisma.client.findMany({
    where: { status: { not: "archived" } }
  });

  const clientMetrics: ClientMetrics[] = [];

  for (const client of clients) {
    // Switch to client database
    await prismaManager.switchDatabase(client.databasePath, client.id);
    const clientPrisma = prismaManager.client;

    // Collect metrics
    const metrics = await collectClientMetrics(clientPrisma, client);
    clientMetrics.push(metrics);
  }

  // Sort and aggregate
  return {
    clients: sortClients(clientMetrics, sortBy),
    totals: aggregateTotals(clientMetrics)
  };
}
```

Evidence-Ready Audit Package Generator

Audit Package Service:

```
export async function createAuditPackageZip(
  prisma: PrismaClient,
  options: AuditPackageOptions
): Promise<Buffer> {
  const archive = archiver("zip", { zlib: { level: 9 } });

  // Collect all data
  const data = await collectAuditPackageData(prisma, options);

  // Create folder structure
  const folders = [
    "Executive-Report",
    "Technical-Report",
    "Evidence",
    "Framework-Mapping",
    "Assets",
    "Agents",
    "Raw-Data",
    "AI-Insights"
  ];

  // Add framework-specific evidence folders
  for (const framework of data.frameworks) {
    archive.directory(`Evidence/${framework.code}/`, false);
  }

  // Add all evidence files
  for (const evidence of data.evidenceFiles) {
    archive.file(evidence.path, { name: `Evidence/${evidence.framework}/${
  }

  // Add reports
  archive.append(JSON.stringify(data.frameworkSummaries, null, 2), {
    name: "Framework-Mapping/summaries.json"
  });

  return archive.finalize();
}
```

Compliance Digital Twin Service

Digital Twin Implementation:

```
export async function generateDigitalTwinSnapshot(
  prisma: PrismaClient
): Promise<DigitalTwinSnapshot> {
  // 1. Calculate current compliance state
  const currentState = await calculateCurrentComplianceState(prisma);

  // 2. Analyze control aging
  const controlAging = await analyzeControlAging(prisma);

  // 3. Predict future scores
  const predictedScores7d = await predictComplianceScore(prisma, 7);
  const predictedScores30d = await predictComplianceScore(prisma, 30);
  const predictedScores90d = await predictComplianceScore(prisma, 90);

  // 4. Forecast control degradation
  const controlDegradation = await forecastControlDegradation(prisma, cont

  // 5. Analyze risk drift
  const riskDrift = await analyzeRiskDrift(prisma);

  // 6. Generate AI forecast
  const aiForecast = await generateAIForecast(
    prisma,
    currentState,
    controlAging,
    predictedScores7d,
    predictedScores30d,
    predictedScores90d,
    controlDegradation,
    riskDrift
  );

  // 7. Store snapshot
  return await prisma.digitalTwinSnapshot.create({
    data: {
      currentComplianceScore: currentState.overallCompliance,
      frameworkScores: currentState.frameworkScores,
      controlAging: controlAging as any,
      predictedScores7d: predictedScores7d as any,
      predictedScores30d: predictedScores30d as any,
      predictedScores90d: predictedScores90d as any,
      controlDegradation: controlDegradation as any,
      riskDrift: riskDrift as any,
      aiForecast
    }
  });
}
```

Explainable AI Service

AI Explanation Architecture:

```
export async function generateExplanation(
  prisma: PrismaClient,
  request: ExplanationRequest
): Promise<ExplanationResult> {
  let result: ExplanationResult;

  switch (request.itemType) {
    case "score":
    case "weighted_score":
      result = await explainWeightedScore(prisma, request.itemId, request.
      break;
    case "anomaly":
      result = await explainAnomaly(prisma, request.itemId, request.contex
      break;
    case "zero_trust":
      result = await explainZeroTrust(prisma, request.itemId, request.cont
      break;
    case "digital_twin":
      result = await explainDigitalTwin(prisma, request.itemId, request.cc
      break;
  }

  // Save to audit log
  await prisma.aIExplanation.create({
    data: {
      itemId: request.itemId,
      itemType: request.itemType,
      explanation: result.explanation,
      reasoning: result.reasoning as any,
      confidenceRating: result.reasoning.confidenceRating
    }
  });

  return result;
}

// Explanation structure
interface ExplanationResult {
  explanation: string;
  reasoning: {
    whyScore: string;
    dataInfluences: Array<{ source: string; weight: number; impact: string
    controlDependencies: Array<{ controlId: string; relationship: string }
    frameworkRelationships: Array<{ framework: string; relevance: string }
    historicalTrends: Array<{ date: string; score: number; change: string
    confidenceRating: number;
    confidenceFactors: string[];
  };
}
```

Vendor Contract Analysis Service

NLP Contract Analysis:

```
export async function analyzeContract(
  prisma: PrismaClient,
  contractText: string,
  contractName: string,
  vendorName: string
): Promise<ContractAnalysisResult> {
  const systemPrompt = `You are an expert contract analyst. Analyze vendor
  1. Key clauses and categories
  2. Security obligations
  3. High-risk sections
  4. Framework mappings
  5. Recommended controls
  6. Risk assessment`;

  const userPrompt = `Analyze this contract:
  Contract: ${contractName}
  Vendor: ${vendorName}
  Text: ${contractText.substring(0, 150000)}`;

  const analysis = await callAI(userPrompt, systemPrompt);

  return {
    clauses: analysis.clauses,
    securityObligations: analysis.securityObligations,
    highRiskSections: analysis.highRiskSections,
    frameworkMappings: analysis.frameworkMappings,
    recommendedControls: analysis.recommendedControls,
    riskScore: calculateRiskScore(analysis),
    riskLevel: scoreToRiskLevel(analysis.riskScore)
  };
}
```

Dynamic Scenario Injection Service

Scenario Injection Architecture:

```
export async function applyScenarioInjection(
  prisma: PrismaClient,
  scenarioConfig: ScenarioConfig,
  name: string
): Promise<ScenarioInjectionResult> {
  // 1. Capture baseline state
  const baselineState = await captureBaselineState(prisma);

  // 2. Convert to What-If simulation input
  const whatIfInput = convertScenarioToWhatIfInput(scenarioConfig, baselin

  // 3. Run What-If simulation
  const simulationResult = await runWhatIfSimulation(prisma, whatIfInput);

  // 4. Calculate simulated state
  const simulatedState = calculateSimulatedState(baselineState, simulation

  // 5. Calculate state diff
  const stateDiff = calculateStateDiff(baselineState, simulatedState);

  // 6. Generate AI impact forecast
  const aiImpactForecast = await generateAIImpactForecast(
    scenarioConfig,
    baselineState,
    simulatedState,
    stateDiff,
    simulationResult
  );

  // 7. Save scenario injection
  return await prisma.scenarioInjection.create({
    data: {
      name,
      scenarioConfig: scenarioConfig as any,
      baselineState: baselineState as any,
      simulatedState: simulatedState as any,
      stateDiff: stateDiff as any,
      aiImpactForecast: aiImpactForecast as any
    }
    });
}
```

What-If Simulation Engine

Simulation Engine Architecture:

```
export async function runWhatIfSimulation(
  prisma: PrismaClient,
  input: WhatIfScenarioInput
): Promise<WhatIfSimulationResult> {
  // Get baseline data
  const baseline = await getBaselineData(prisma, input);

  // Run domain-specific simulation
  let projectedScores: ProjectedScores;
  let componentChanges: ComponentChange[];

  switch (input.domain) {
    case "framework_compliance":
      const frameworkResult = await simulateFrameworkChanges(prisma, input
      projectedScores = frameworkResult.scores;
      componentChanges = frameworkResult.changes;
      break;

    case "risk_register":
      const riskResult = await simulateRiskChanges(prisma, input, baseline
      projectedScores = riskResult.scores;
      componentChanges = riskResult.changes;
      break;

    case "zero_trust_baseline":
      const zeroTrustResult = await simulateZeroTrustChanges(prisma, input
      projectedScores = zeroTrustResult.scores;
      componentChanges = zeroTrustResult.changes;
      break;
  }

  // Generate AI summary
  const aiContent = await generateAIContent(input, projectedScores, compon

  return {
    scenarioId: input.scenarioId || `scenario_${Date.now()}`,
    inputs: input,
    projectedScores,
    componentChanges,
    aiSummary: aiContent.summary,
    executiveNarrative: aiContent.executiveNarrative,
    recommendations: aiContent.recommendations
  };
}
```

Zero-Trust Architecture

Zero-Trust Calculation Engine:

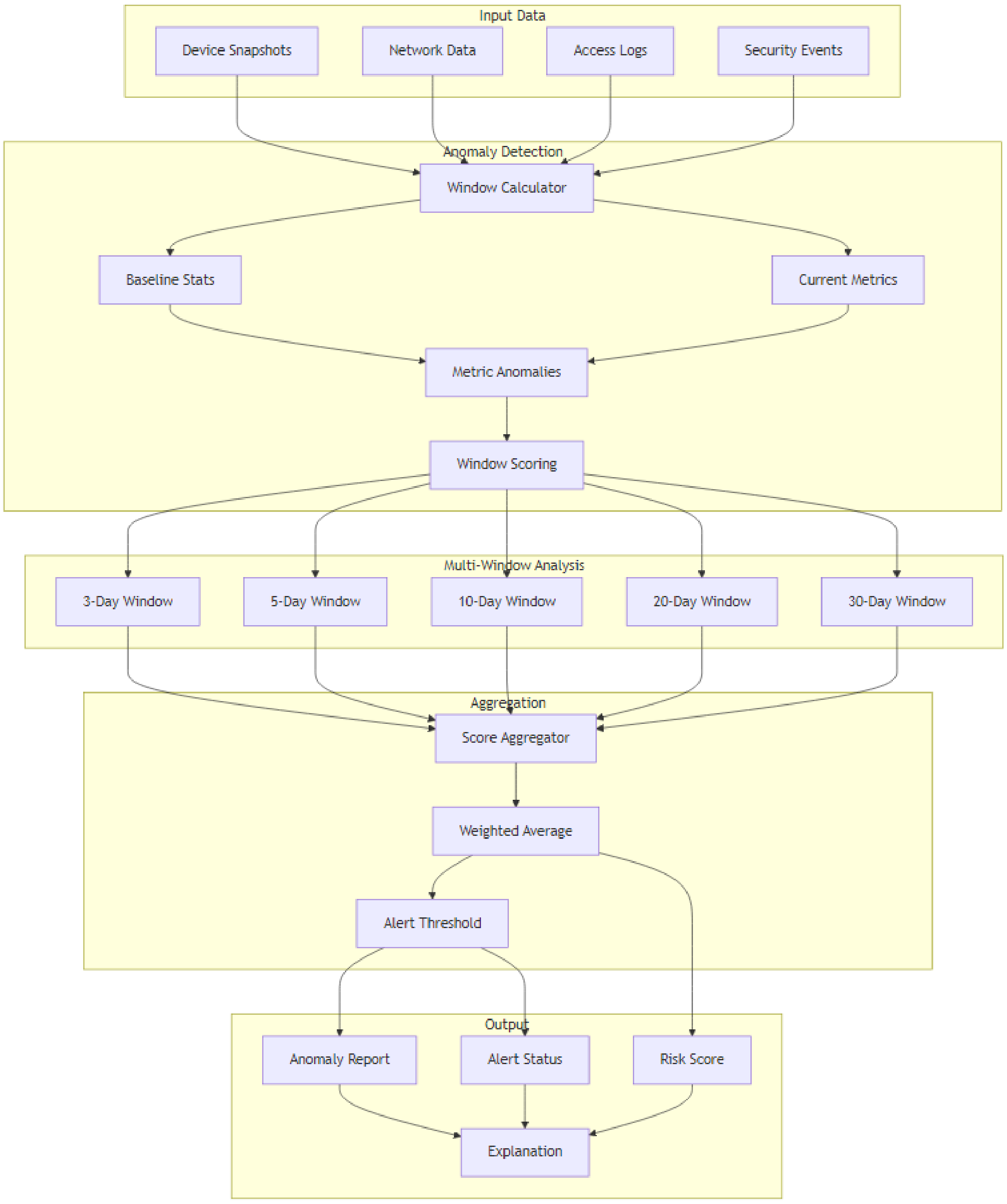
```
export async function calculateOverallZeroTrust(
  prisma: PrismaClient,
  assetId?: string
): Promise<ZeroTrustOverallResult> {
  const dimensions = [
    "IDENTITY_ACCESS",
    "DEVICE_TRUST",
    "NETWORK_SEGMENTATION",
    "APPLICATION_TRUST",
    "DATA_GOVERNANCE",
    "AUTOMATION_RESPONSE"
  ];

  // Calculate scores for each dimension
  const dimensionScores = await Promise.all(
    dimensions.map(dim => calculateDimensionScore(prisma, dim, assetId))
  );

  // Calculate weighted overall score
  let totalWeightedScore = 0;
  let totalWeight = 0;

  for (const result of dimensionScores) {
    const weight = result.weight || 1.0;
    totalWeightedScore += result.score * weight;
    totalWeight += weight;
  }

  const overallScore = totalWeight > 0 ? totalWeightedScore / totalWeight
  const maturity = scoreToMaturityLevel(overallScore);

  return {
    overallScore,
    maturity,
    dimensionScores,
    recommendations: aggregateRecommendations(dimensionScores)
  };
}
```

Least Privilege Engine:

```
export async function analyzeLeastPrivilege(
  prisma: PrismaClient,
  entityId: string
): Promise<LeastPrivilegeAnalysis> {
  const baseline = await getAccessBaseline(prisma, entityId);
  const currentPermissions = await getCurrentPermissions(prisma, entityId)

  // Identify excessive permissions
  const excessivePermissions = currentPermissions.filter(perm =>
    !baseline.permissions.some(bp => matchesPermission(bp, perm))
  );

  // Identify unused permissions
  const unusedPermissions = baseline.permissions.filter(bp =>
    !hasBeenUsed(bp, entityId)
  );

  // Calculate privilege escalation likelihood
  const escalationLikelihood = calculateEscalationLikelihood(
    excessivePermissions,
    currentPermissions
  );

  return {
    excessivePermissions,
    unusedPermissions,
    privilegeEscalationLikelihood: escalationLikelihood,
    recommendedActions: generateRecommendations(excessivePermissions, unus
    score: calculateLeastPrivilegeScore(excessivePermissions, unusedPermis
  };
}
```

Cybersecurity Reasoning Engine

Anomaly Detection Architecture:



```typescript export function computeAnomalyReport( deviceId: string, snapshots: DeviceSnapshot[], nowTs: Timestamp, config: EngineConfig = DEFAULT_CONFIG ): FinalAnomalyReport { const sorted = snapshots.slice().sort((a, b) => a.timestamp - b.timestamp);
```

const windows = config.windowsDays || [3, 5, 10, 20, 30]; const windowResults: WindowResult[] = [];

for (const days of windows) { const wr = computeWindowResult(sorted, days, nowTs, config); windowResults.push(wr); }

const aggregatedScore = aggregateAcrossWindows(windowResults, config); const alert = aggregatedScore >= (config.alertThreshold ?? 70);

return { deviceId, generatedAt: nowTs, windowResults, aggregatedScore, alert, explanation: generateExplanation(windowResults) }; }

// Window result calculation function computeWindowResult(snapshots: DeviceSnapshot[], lookbackDays: number, nowTs: Timestamp, config: EngineConfig): WindowResult { const baseline = calcWindowBaselineStats(snapshots, lookbackDays, nowTs); const current = calcCurrentMetricsFromSnapshots(snapshots, nowTs); const metricAnomalies = calcMetricAnomalies(baseline, current, config); const windowScore = scoreWindow(metricAnomalies, config);

return { windowDays: lookbackDays, baseline, current, metricAnomalies, windowScore }; }

~~~
### AI Chart Generation Service

**Chart Generation Architecture**:
```typescript
export async function generateChart(
  prisma: PrismaClient,
  request: ChartGenerationRequest
): Promise<ChartSpec> {
  // Fetch data
  const rawData = await fetchChartData(prisma, request);

  // Suggest chart type if not provided
  let chartType = request.preferredChartType;
  if (!chartType) {
    chartType = await suggestChartType(rawData, request.context || "");
  }

  // Generate chart spec with AI
  const systemPrompt = `You are a data visualization expert. Generate a cc
  specification in JSON format for GRC compliance data.`;

  const prompt = `Generate a ${chartType} chart:
  Data: ${JSON.stringify(rawData.data, null, 2)}
  Context: ${request.context || "GRC compliance dashboard"}`;

  const aiResponse = await callModel(prompt, systemPrompt);

  return {
    type: chartType,
    title: aiResponse.title,
    description: aiResponse.description,
    data: transformData(rawData, chartType),
    summary: aiResponse.summary,
    colors: aiResponse.colors || getDefaultColors(chartType)
  };
}
~~~

AI Report Editor Service

Report Optimization Architecture:

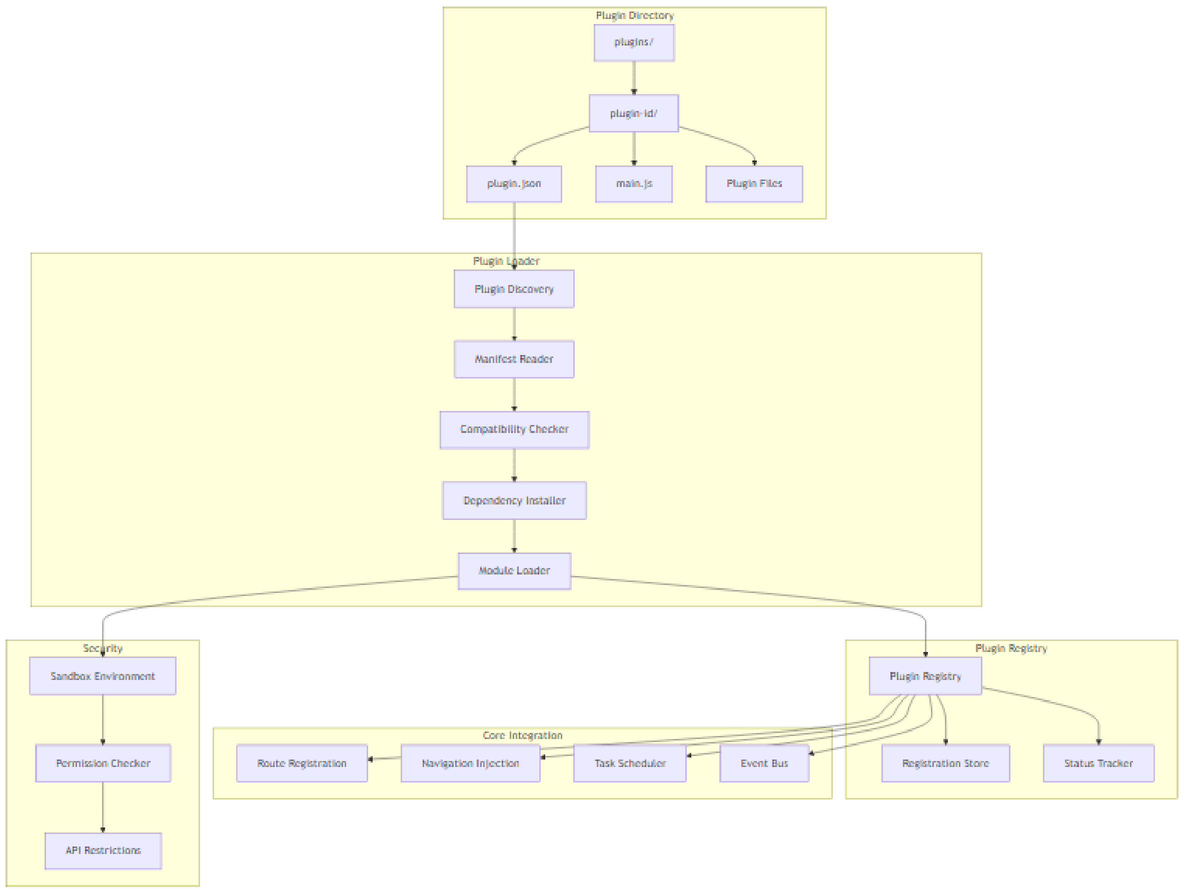
```
export async function optimizeSection(
  sectionText: string,
  tone: Tone = "formal",
  style: Style = "detailed"
): Promise<string> {
  const toneInstructions: Record<Tone, string> = {
    formal: "Use formal, professional language suitable for official compl
    executive: "Use clear, executive-friendly language. Focus on business 
    technical: "Use precise technical terminology with detailed explanatic
    narrative: "Use a narrative, storytelling style that flows naturally."
    concise: "Be extremely concise and to the point."
  };

  const systemPrompt = `You are an AI assistant specialized in GRC reporti
  ${toneInstructions[tone]}
  Optimize report sections for clarity, grammar, style, and flow.`;

  const prompt = `Optimize the following report section:
  ${sectionText}
  Tone: ${tone}
  Style: ${style}`;

  return await callModel(prompt, systemPrompt);
}
```

Plugin System Architecture

Overview

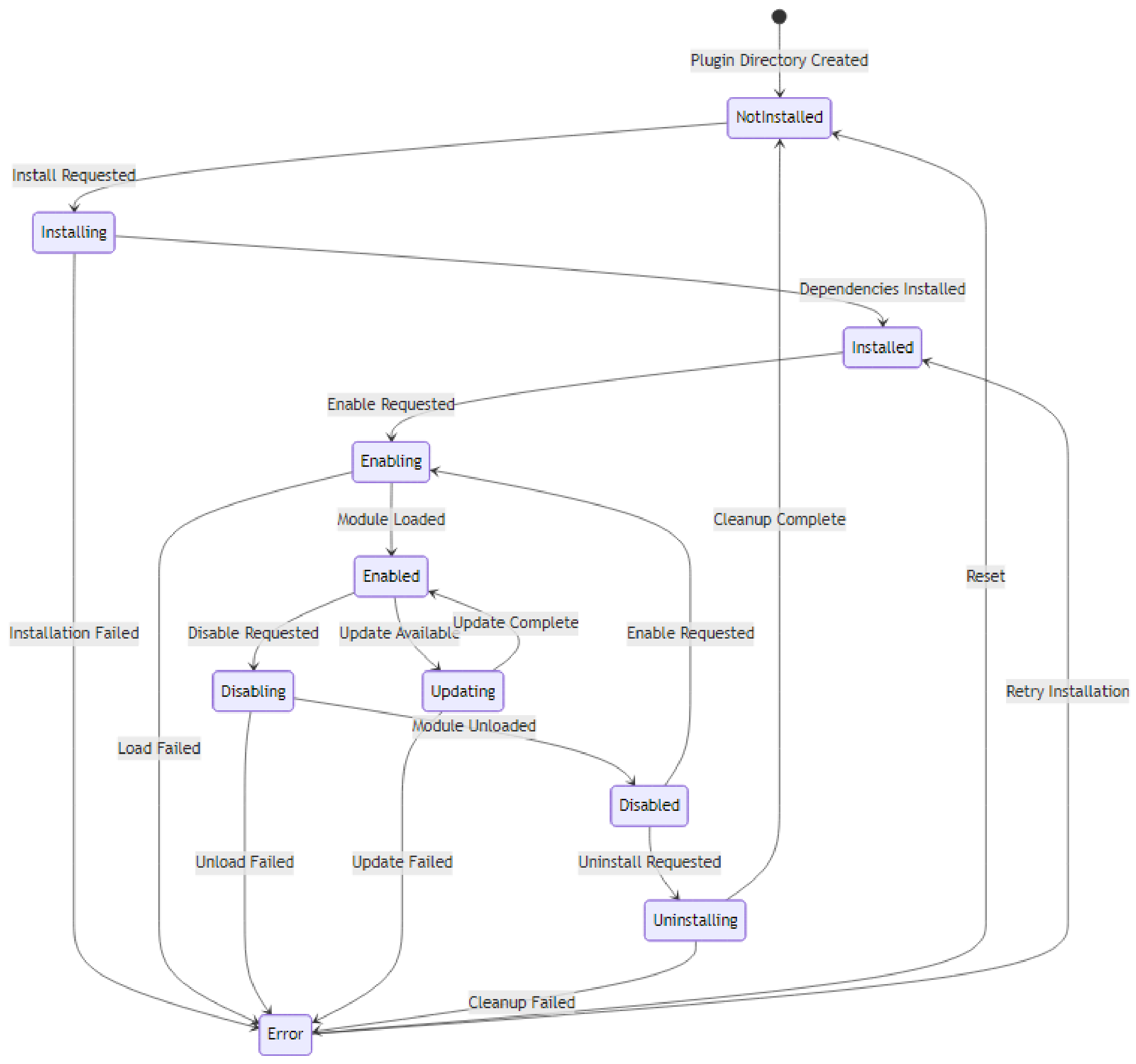
RiskFortress includes a fully modular plugin system that allows extending functionality without modifying core application code. The plugin system supports multiple runtime environments (Node.js, Python, Go) and provides secure sandboxing with tenant isolation.

Plugin System Overview

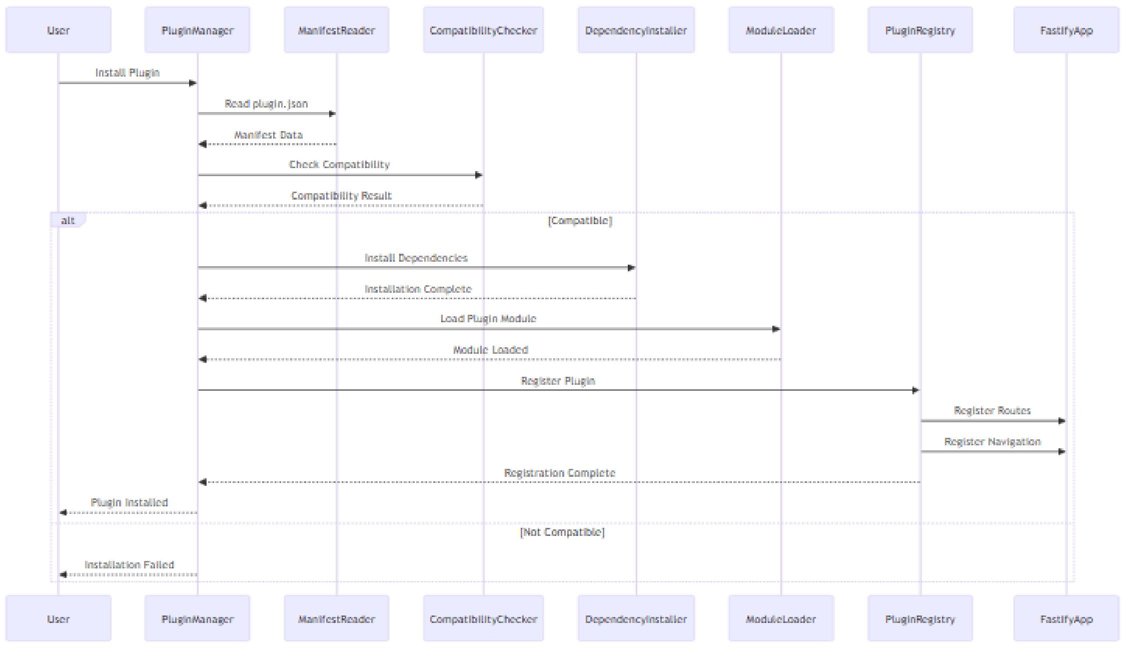


Plugin Lifecycle

Plugins go through a well-defined lifecycle from installation to activation:

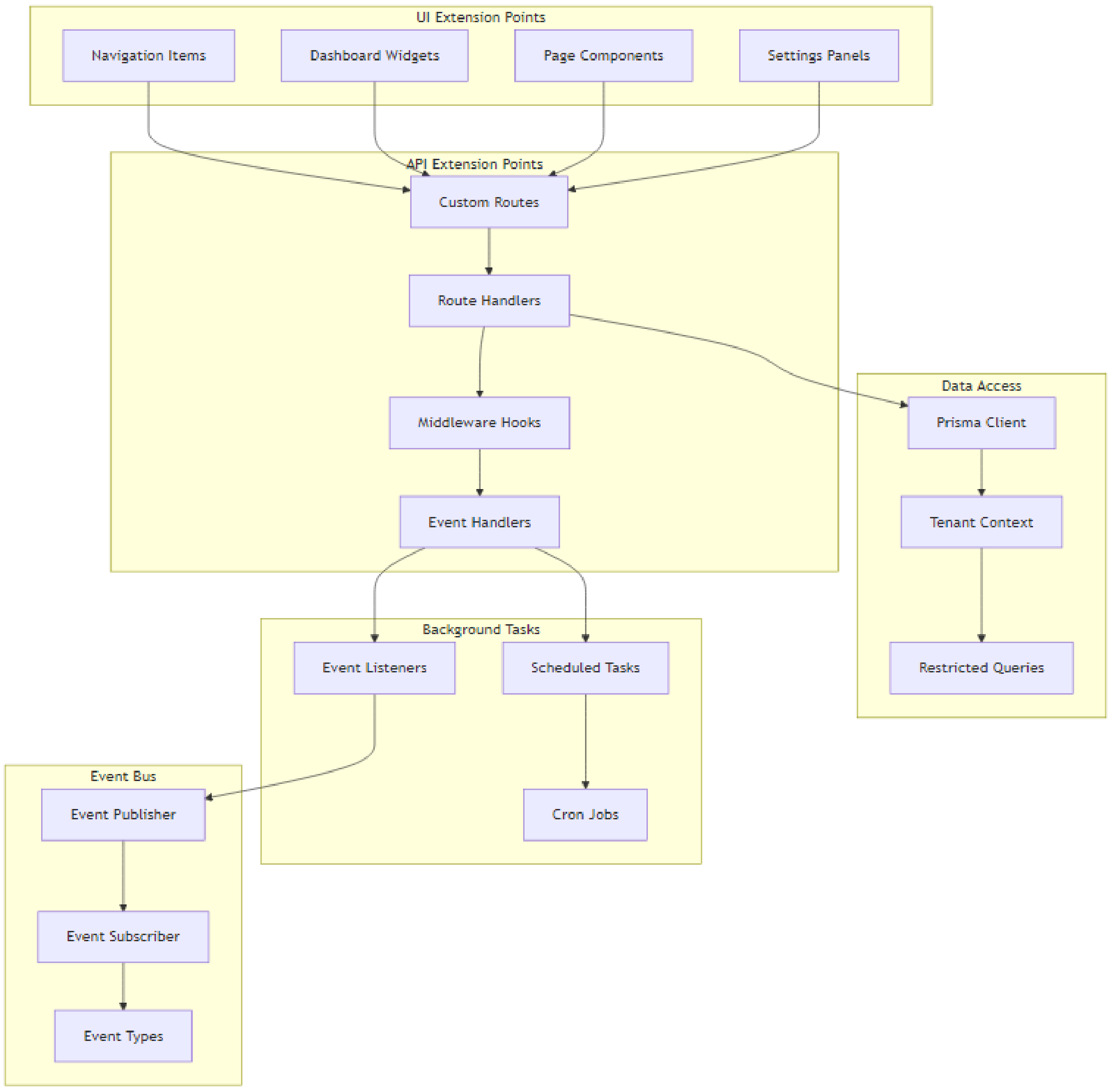

Plugin Directory Created
NotInstalled
Install Requested
Installing
Dependencies Installed
Installed
Enable Requested
Enabling
Module Loaded
Enabled
Cleanup Complete
Reset
Installation Failed
Disable Requested
Update Available
Update Complete
Enable Requested
Disabling
Updating
Retry Installation
Module Unloaded
Load Failed
Disabled
Unload Failed
Update Failed
Uninstall Requested
Uninstalling
Cleanup Failed
Error

Plugin Registration Flow



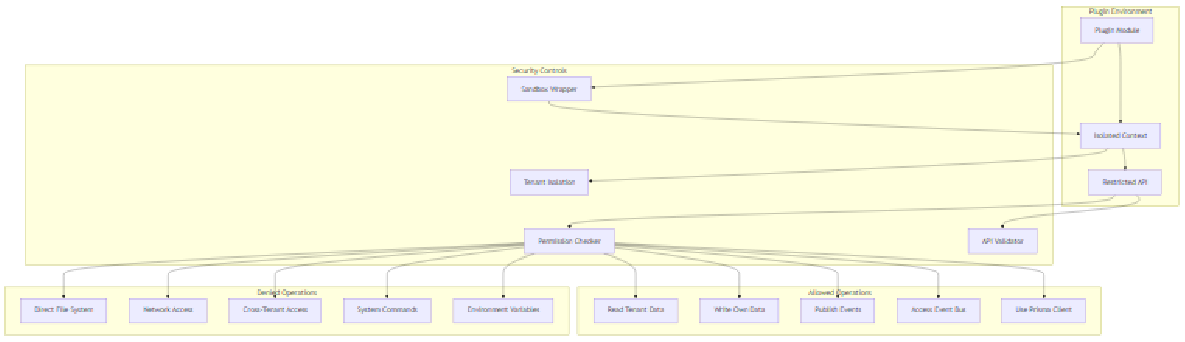
Plugin API Surface

Plugins can extend the platform through multiple extension points:



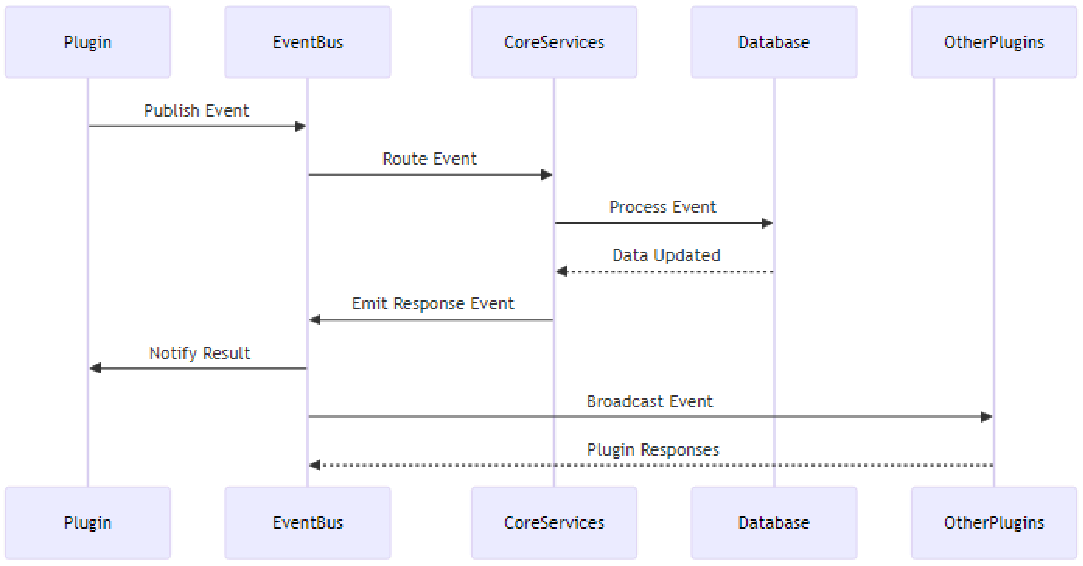
Plugin Sandbox & Security Model

Plugins run in isolated sandbox environments with restricted access:



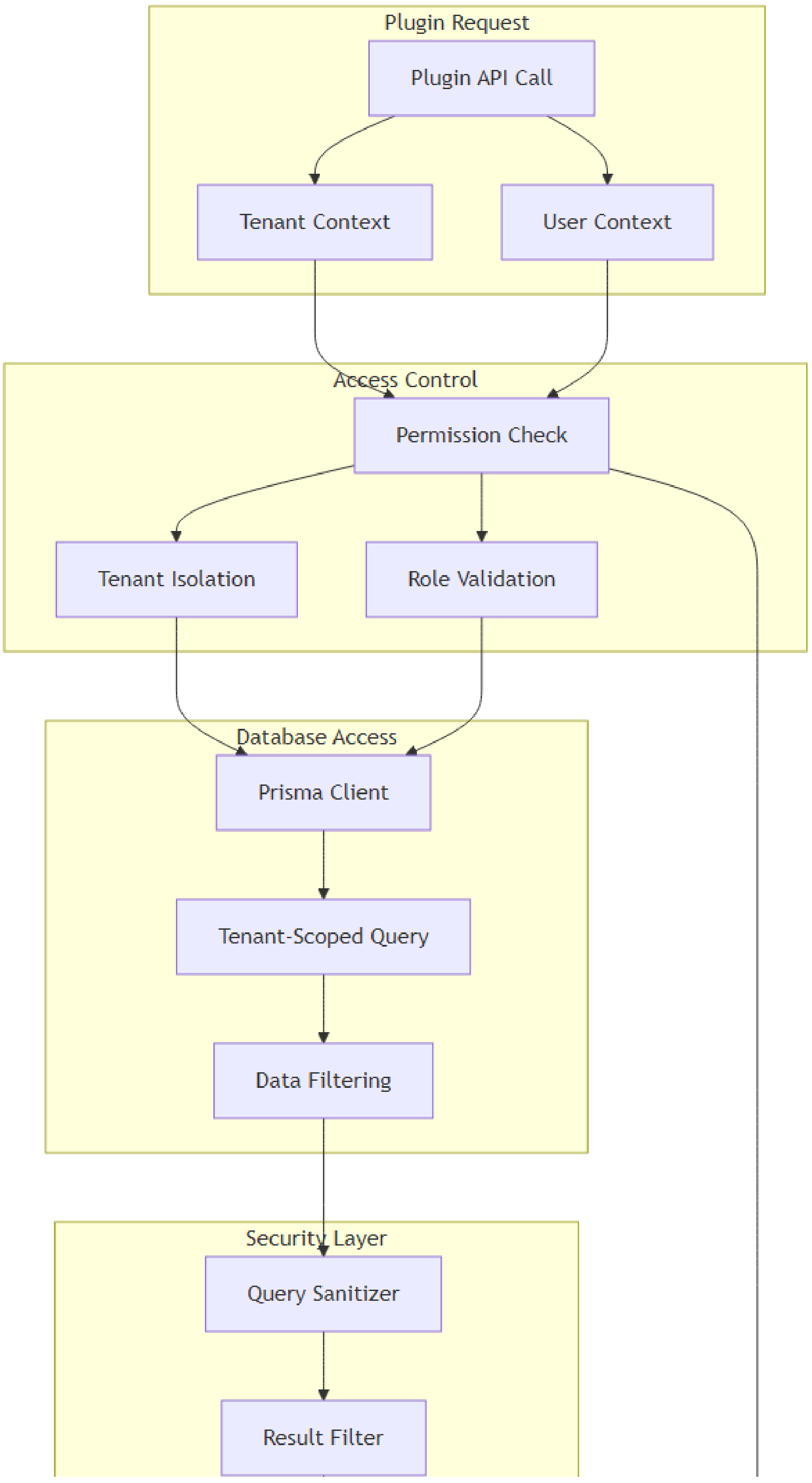
Plugin Event Flow

Plugins communicate with the core system and other plugins through an event bus:



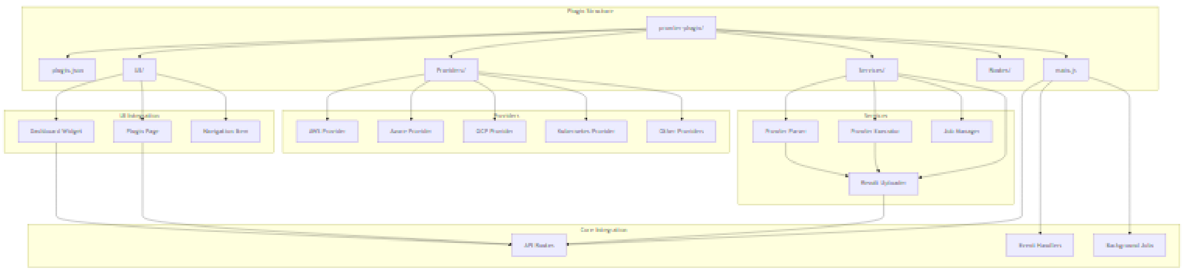
Plugin Data Access Model

Plugins access data through a secure, tenant-isolated model:


Plugin Request
Plugin API Call
Tenant Context
User Context
Access Control
Permission Check
Tenant Isolation
Role Validation
Database Access
Prisma Client
Tenant-Scoped Query
Data Filtering
Security Layer
Query Sanitizer
Result Filter

Example: Prowler Plugin Architecture

The Prowler plugin demonstrates a complete plugin implementation:



Plugin Manifest Structure

Plugins are defined by a `plugin.json` manifest file:

```
interface PluginManifest {
  id: string;                       // Unique plugin identifier
  name: string;                      // Display name
  version: string;                   // Semantic version
  description: string;              // Plugin description
  author?: string;                  // Author information
  license?: string;                 // License type
  icon?: string;                    // Icon URL or path
  entry: string;                    // Entry point file
  nav?: NavigationItem[];           // Navigation items
  requires?: {                       // Runtime requirements
    python?: string;
    node?: string;
    go?: string;
  };
  dependencies?: {                   // Dependency files
    python?: string;                // requirements.txt
    node?: string;                   // package.json
    go?: string;                     // go.mod
  };
  env?: string[];                    // Required environment variables
  security?: {                       // Security configuration
    permissions?: string[];
    networkAccess?: boolean;
    fileSystemAccess?: boolean;
  };
  integrations?: {                    // Framework/module integrations
    frameworks?: string[];
    modules?: string[];
  };
}
```

Plugin API Endpoints

The platform provides RESTful API endpoints for plugin management:

- `GET /api/plugins` - List all plugins
- `GET /api/plugins/:id` - Get plugin details

- `GET /api/plugins/:id/compatibility` - Check compatibility
- `POST /api/plugins/:id/install` - Install plugin
- `POST /api/plugins/:id/uninstall` - Uninstall plugin
- `POST /api/plugins/:id/enable` - Enable plugin
- `POST /api/plugins/:id/disable` - Disable plugin
- `POST /api/plugins/:id/reload` - Reload plugin
- `GET /api/plugins/:id/logs` - Get plugin logs

Deployment Architecture

Development Setup

```
# Install dependencies
npm install

# Setup environment
cp .env.example .env
# Configure DATABASE_URL, JWT_SECRET, ENCRYPTION_KEY

# Run migrations
npx prisma migrate dev

# Seed database
npx prisma db seed

# Start development servers
npm run dev
# Runs Next.js (port 3000) and Fastify (port 3001) concurrently
```

Production Deployment

Docker Deployment:

```
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
RUN npx prisma generate

FROM node:18-alpine
WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/server ./server
COPY --from=builder /app/prisma ./prisma
COPY --from=builder /app/package.json ./
EXPOSE 3000
CMD ["npm", "start"]
```

Environment Variables:

```
DATABASE_URL=postgresql://user:pass@host:5432/db
JWT_SECRET=your-secret-key
ENCRYPTION_KEY=your-encryption-key
OPENAI_API_KEY=your-openai-key  # Optional
OLLAMA_BASE_URL=http://ollama:11434  # Optional
NODE_ENV=production
```

Kubernetes Deployment

Deployment YAML:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: riskfortress
spec:
  replicas: 3
  selector:
    matchLabels:
      app: riskfortress
  template:
    metadata:
      labels:
        app: riskfortress
    spec:
      containers:
      - name: app
        image: riskfortress:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: riskfortress-secrets
              key: database-url
```

Performance & Scalability

Database Optimization

- **Indexes**: Strategic indexes on foreign keys and frequently queried fields
- **Connection Pooling**: Prisma connection pooling for efficient database connections
- **Query Optimization**: Use `select` to limit fields, `include` for relations
- **Pagination**: All list endpoints support pagination

Caching Strategy

- **TanStack Query**: Automatic caching of API responses
- **Next.js Caching**: Built-in caching for static pages
- **Database Query Caching**: Consider Redis for frequently accessed data

Scalability Considerations

- **Horizontal Scaling**: Stateless API allows multiple instances
- **Database Scaling**: PostgreSQL supports read replicas
- **CDN**: Static assets served via CDN
- **Load Balancing**: Distribute traffic across instances

Appendix: Technical Specifications

System Requirements

Server:

- Node.js 18+ (LTS recommended)
- 4GB RAM minimum, 8GB recommended
- 10GB disk space
- PostgreSQL 12+ (production) or SQLite (development)

Client:

- Modern browser (Chrome 90+, Firefox 88+, Safari 14+)
- JavaScript enabled
- 2GB RAM recommended

Technology Versions

Frontend:

- Next.js: 15.5.7

- React: 18.3.1
- TypeScript: 5.6.3
- TailwindCSS: 3.4.14

Backend:

- Fastify: 5.0.0
- Prisma: 6.19.0
- Zod: 3.23.8

Database:

- PostgreSQL: 12+ (recommended)
- SQLite: 3.x (development)

API Rate Limits

- Authentication: 10 requests/minute
- General API: 100 requests/minute
- Report Generation: 5 requests/minute
- AI Endpoints: 20 requests/minute

File Size Limits

- Logo Upload: 1MB
- Import Files: 50MB
- Report Generation: No limit (memory dependent)

This technical white paper is intended for developers, architects, and IT operations teams. For business-focused documentation, please refer to the Executive White Paper.