**ChainNotary**

**Project Architecture**

**📋 Architecture Overview**

ChainNotary is a decentralized document verification, XBRL processing, and AI analytics platform built on the Internet Computer Protocol (ICP). It features a comprehensive multi-tenant architecture with advanced user roles, institution management, and real-time notifications through Discord and email integrations.

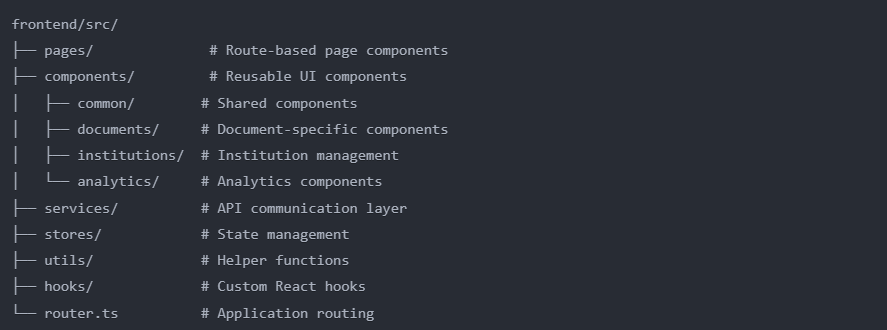
**System Architecture Layers**

**1. Frontend Layer (React + TypeScript)**

**Technology Stack:**

* **Framework: React 19.1.0 with TypeScript**
* **Build Tool: Vite 5.4.11**
* **UI Framework: Ant Design 5.26.7**
* **Styling: Tailwind CSS 4.1.11**
* **Icons: Lucide React**
* **Routing: React Router 7.7.1**
* **State Management: Custom stores (authStore, documentStore, institutionStore)**
* **ICP Integration: @dfinity/\* packages**

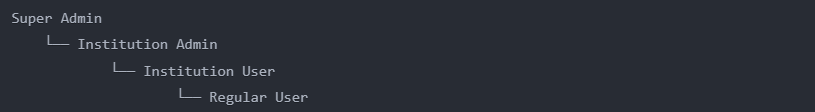
**Architecture Pattern:**



**Key Components:**

* **Pages: Dashboard, DocumentUpload, DocumentDetails, Analytics, Institution Management, User Profile**
* **Services: Document operations, Analytics API, User management, Institution services**
* **Utils: File processing, hash calculation, form validation, timestamp helpers**
* **Stores: Authentication, Document, Institution, Notification management**

**2. User Types & Role Hierarchy**

****

**Key Components:**

* **Super Admin: System-wide access, institution management, global analytics**
* **Institution Admin: Institution-level management, user invitations, document management**
* **Institution User: Document upload/verification, limited analytics, XBRL processing**
* **Regular User: Basic document verification, profile management**

**3. ICP Blockchain Layer**

**Configuration (dfx.json):**

****

**Key Features:**

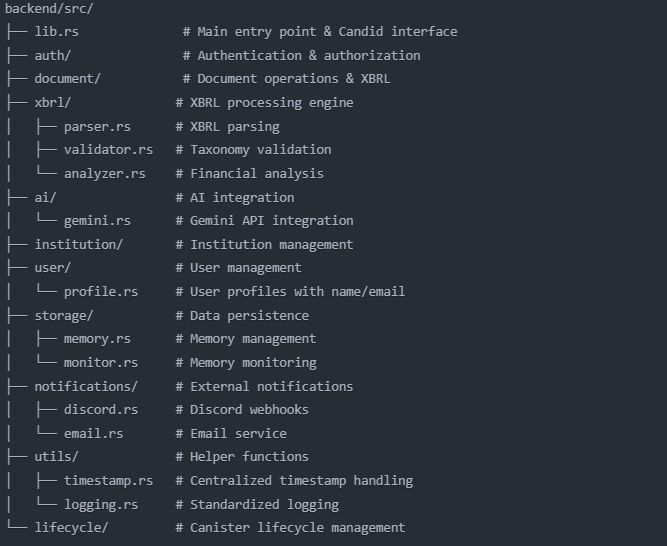
* **Internet Identity: Decentralized authentication**
* **Stable Memory: Persistent storage across upgrades**
* **HTTP Outcalls: Non-replicated calls for external services**
* **Candid Interface: Type-safe frontend/backend communication**
* **Memory Monitoring: Hourly wipe detection and tracking**

**4. Backend Canister (Rust + WASM)**

**Technology Stack:**

* **Language: Rust 2021 Edition**
* **Target: wasm32-unknown-unknown (WebAssembly)**
* **Framework: IC-CDK 0.18.3**
* **Storage: IC-Stable-Structures 0.6.7 with Bincode serialization**
* **XBRL Processing: Custom parser with taxonomy validation**
* **PDF Processing: lopdf 0.32**
* **Cryptography: sha2, hex**
* **Serialization: Bincode for efficient storage**

**Module Architecture:**



**Core Data Models:**

**// User with enhanced profile**

**#[derive(CandidType, Serialize, Deserialize)]**

**pub struct User {**

**pub id: Principal,**

**pub role: UserRole,**

**pub profile: UserProfile,**

**pub institution: Option<String>,**

**pub created\_at: u64,**

**pub status: UserStatus,**

**}**

**#[derive(CandidType, Serialize, Deserialize)]**

**pub struct UserProfile {**

**pub name: String,**

**pub email: String,**

**pub avatar\_url: Option<String>,**

**pub metadata: HashMap<String, String>,**

**}**

**// Institution with hierarchy**

**#[derive(CandidType, Serialize, Deserialize)]**

**pub struct Institution {**

**pub id: String,**

**pub name: String,**

**pub admins: Vec<Principal>,**

**pub users: Vec<Principal>,**

**pub settings: InstitutionSettings,**

**pub documents: Vec<String>,**

**pub created\_at: u64,**

**}**

**// Document with XBRL support**

**#[derive(CandidType, Serialize, Deserialize)]**

**pub struct Document {**

**pub id: String,**

**pub hash: String,**

**pub metadata: DocumentMetadata,**

**pub xbrl\_data: Option<XbrlMetadata>,**

**pub encrypted\_content: Vec<u8>,**

**pub status: DocumentStatus,**

**}**

**// XBRL metadata**

**#[derive(CandidType, Serialize, Deserialize)]**

**pub struct XbrlMetadata {**

**pub taxonomy\_ref: String,**

**pub contexts: Vec<XbrlContext>,**

**pub facts: Vec<XbrlFact>,**

**pub validation\_result: ValidationResult,**

**}**

**5. AI Integration Layer**

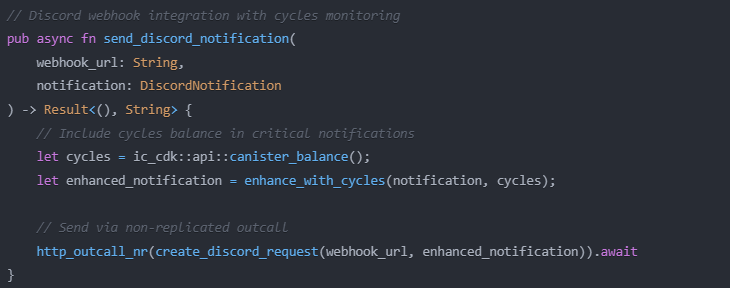
**Gemini Integration**

****

**Features:**

* **PDF Text Extraction**: lopdf library processes binary data
* **AI API Integration**: HTTP outcalls to Gemini 2.5 Pro
* **Multi-Analysis Types**: Financial summary, investment insights, charts
* **Smart Prompting**: Context-aware AI instructions

**Discord Integration**

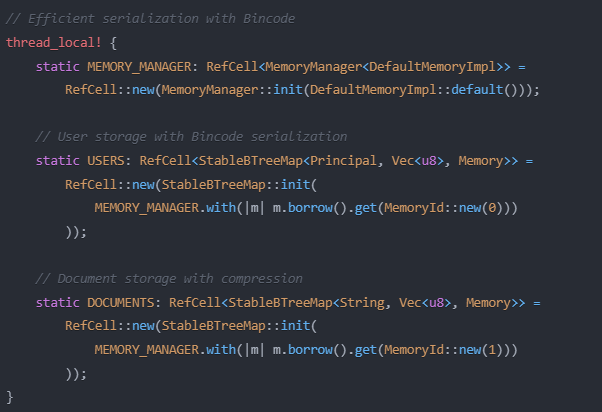
****

**Features:**

* **Monitoring**: checking cycle balance.
* **Notification**: sending notifications every 24 hour period.

**6. Data Storage Architecture**

**Optimized Storage with Bincode:**

****

**Memory Monitoring & Wipe Detection:**

**  
  
Storage Strategy:**

* **Stable Memory**: Survives canister upgrades
* **BTreeMap**: Efficient key-value storage
* **Serialization**: Candid + Serde for type safety
* **Memory Management**: Separate memory regions per data type

**Data Flow Architecture**

**User Registration Flow:**

1. User visits site → Frontend

2. Internet Identity authentication → authenticate()

3. Check existing user → whoami()

4. New user registration → create\_user\_profile()

5. Set name and email → update\_profile()

6. Institution assignment (if applicable) → assign\_to\_institution()

7. Welcome notification → send\_discord\_notification()

8. Dashboard redirect → user\_dashboard()

**Document Upload with XBRL:**

1. User uploads document → Frontend validation

2. Detect document type → check\_xbrl\_format()

3. Calculate hash → sha256\_hash()

4. Backend upload → upload\_document()

5. XBRL validation (if applicable) → validate\_xbrl\_taxonomy()

6. AI analysis → analyze\_document\_with\_ai()

7. Bincode serialization → serialize\_with\_bincode()

8. Store in stable memory → DOCUMENTS.insert()

9. Update user history → add\_to\_document\_history()

10. Discord notification → notify\_document\_uploaded()

11. Return document ID → Frontend redirect

**Institution Management Flow:**

1. Admin creates institution → create\_institution()

2. Configure settings → update\_institution\_settings()

3. Invite users → invite\_institution\_member()

4. Email invitation → send\_email\_invite()

5. User accepts → join\_institution()

6. Update hierarchy → update\_user\_role()

7. Grant permissions → assign\_permissions()

8. Analytics update → refresh\_institution\_analytics()

**Security Architecture**

**Authentication & Authorization:**

* **Internet Identity: Decentralized authentication system**
* **Role-Based Access Control (RBAC): Hierarchical permission system**
* **Principal validation: Every operation validates caller identity**
* **Session management: Secure session handling with whoami() function**

**Data Security:**

* **Document encryption: All documents encrypted before storage**
* **Hash verification: SHA-256 integrity checks**
* **XBRL validation: Taxonomy-based validation for financial documents**
* **Audit logging: Comprehensive activity tracking**

**API Security:**

* **Non-replicated outcalls: Prevents consensus issues**
* **HTTPS enforcement: All external calls use HTTPS**
* **API key rotation: Regular key updates for external services**
* **Rate limiting: Protection against abuse**

**CI/CD & Deployment Architecture**



**Deployment Scripts:**

* scripts/build-backend.sh: Optimized Rust build with Bincode
* scripts/deploy-backend.sh: Backend-specific deployment
* scripts/deploy-mainnet.sh: Full mainnet deployment with cycles

**Performance Optimizations**

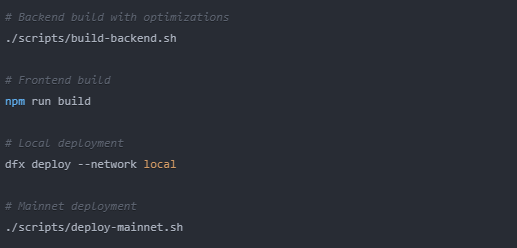
**Storage Optimizations:**

* **Bincode serialization**: 50% smaller than Candid
* **Comprehensive cleanup**: Automatic old document archival
* **Memory monitoring**: Proactive memory management
* **Index optimization**: Fast document queries

**Cycle Usage Optimization:**

* **Logging consolidation**: Reduced logging overhead
* **Heartbeat optimization**: Disabled expensive operations
* **Query optimization**: Removed unused functions
* **Admin monitoring**: get\_cycles\_balance() for tracking

**Deployment Architecture**

****

**Key Development Features:**

* **Hot reload: Vite dev server for frontend**
* **Type safety: End-to-end TypeScript + Rust + Candid**
* **Automated testing: CI/CD pipeline with comprehensive tests**
* **Memory monitoring: Real-time usage tracking**
* **Cycle monitoring: Admin dashboard for cycle balance**

**Architectural Strengths**

**✅ Enhanced Features:**

* **Multi-tenant architecture: Full institution hierarchy support**
* **XBRL compliance: Financial document processing**
* **Real-time notifications: Discord and email integration**
* **Advanced monitoring: Memory and cycles tracking**
* **User profiles: Complete user management system**

**✅ Performance Optimizations:**

* **Bincode serialization: 50% storage reduction**
* **Logging optimization: Reduced cycle usage**
* **Memory management: Proactive cleanup and monitoring**
* **Non-replicated outcalls: Improved external service reliability**

**✅ Developer Experience:**

* **CI/CD pipeline: Automated mainnet deployment**
* **Comprehensive scripts: Build and deployment automation**
* **Type safety: Full-stack type checking**
* **Monitoring tools: Admin dashboard and analytics**

**✅ Security Enhancements:**

* **Role-based access: Hierarchical permission system**
* **Audit trails: Complete activity logging**
* **Encrypted storage: Document encryption at rest**
* **API security: Rate limiting and key rotation**

This architecture provides a robust, scalable foundation for a decentralized document verification platform with XBRL support, AI capabilities, and comprehensive institution management, all while maintaining high security standards and optimal performance on the Internet Computer Protocol.