Boosting MSME digital adoption

TSI DX Node

An open-source, decentralized software solution for peer-to-peer data exchange among business partners

System Development Plan

16th July 2025
V1.0

Prepared By

Satish Ayyaswami
Email: admin@tsicoop.org / satish@tsiconsulting.in
Mobile: 99401 61886
LinkedIn: https://www.linkedin.com/in/satishayyaswami
Substack: https://techadvisory.substack.com

TSI Tech Solutions Cooperative Foundation
A section 8 company

https://tsicoop.org

Table of Contents

1. About Us

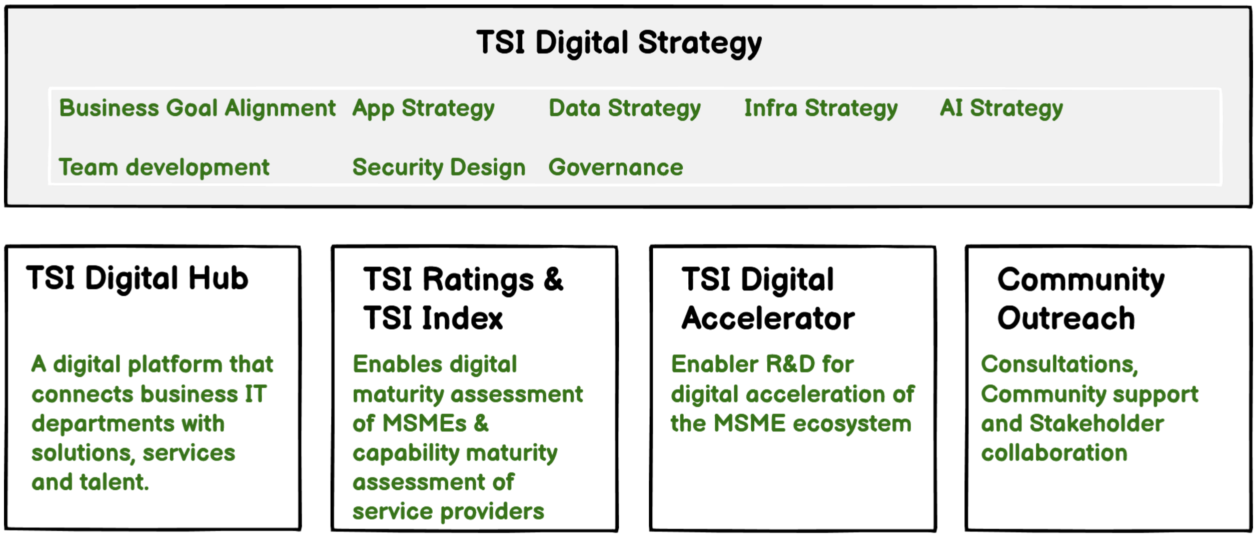
TSI Tech Solutions Cooperative Foundation (TSI Coop) initiative aims to address critical gaps in the MSME digital ecosystem by enabling businesses to implement successful technology strategies with the help of niche providers. Our mission also focuses on creating sustainable pathways for graduates from tier-3 and tier-4 institutions while fostering a more equitable and efficient domestic IT supply chain.

TSI stands for Technology & Social Impact. 'Coop' embodies the spirit of a cooperative economic model, where smaller providers and MSMEs collectively drive digital adoption within the ecosystem, fostering shared growth and opportunity.

While our IT machinery prioritizes foreign markets, GCCs, high profile startups and large enterprises, MSMEs are underserved. We aim to unlock their potential by facilitating the following:

- Discovery of niche providers, products, services and talent. Direct Interactions.
- Continuous digital maturity assessment of MSMEs ecosystem participants, helping stakeholders' identify areas for improvement
- Enabler R&D for digital acceleration of MSME ecosystem
- Community support and stakeholder collaboration

Our offerings for the MSME ecosystem below:



TSI Digital Strategy

Business Goal Alignment, App Strategy, Data Strategy, Infra Strategy, AI Strategy, Team development, Security Design, Governance

TSI Digital Hub

A digital platform that connects business IT departments with solutions, services and talent.

TSI Ratings & TSI Index

Enables digital maturity assessment of MSMEs & capability maturity assessment of service providers

TSI Digital Accelerator

Enabler R&D for digital acceleration of the MSME ecosystem

Community Outreach

Consultations, Community support and Stakeholder collaboration

TSI DX Node is a key component of our TSI Digital Accelerator program.

2. Problem Statement

Every organisation exchanges a lot of data daily with its business partners, investors, auditors, regulators, etc. Most of these are shared via SFTP, emails (sometimes with password protection) and APIs.

Data sharing agreements between organisations usually spell out the data transfer purpose, schema, security, purging and other compliance clauses. However, it is tough to enforce them when the operations team uploads the data files to the SFTP folder or shares them via email. API-based data contracts may be better at enforcement, but they require custom implementation on both sides, which is expensive for most situations.

We need an open-source software solution that lets businesses interface with partners in a standardised manner. The solution should help build a decentralised network for data exchange between partner organisations with built-in data governance support, as described below

- Peer to Peer data transfer between organisations
- The solution runs on the IT infrastructure of each participating business
- Partner Registry on DX Nodes with auto-discovery support
- Data Contract schema definitions as per the signed agreement
- Personally Identifiable Information (PII) Anonymization
- Scripting support for data validations
- Web Application Interface & API endpoints for uploading to & downloading from nodes
- Support for broadcasting the data to multiple partners at the same time
- Secure data-in-transit channel using PKI
- Role based access controls
- Detailed audit logs around each data transfer
- Data archiving & purging support
- Data synchronisation fault tolerance

The solution should be simple, so all organisations can install & benefit. With the open source utility as a base, data management vendors can offer commercial solutions to improve the data exchange experience of medium and large organisations. At the same time, small organisations can continue to use the open source solution.

3. Scope

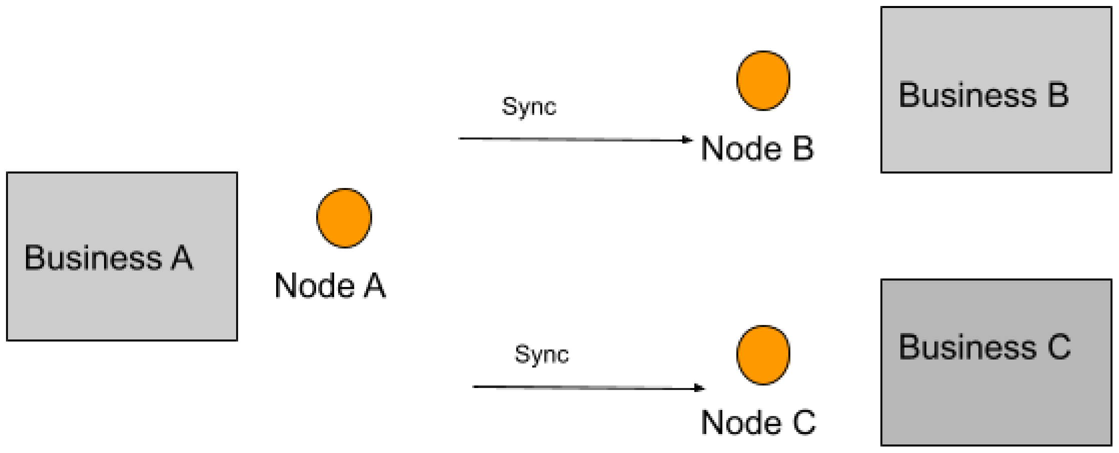
This document outlines the system architecture for the TSI DX Node, an open-source solution designed to facilitate secure, governed, and standardized peer-to-peer data transfer between partner organizations. Each participating organization hosts its own TSI DX Node, enabling direct data exchange while enforcing data contracts, ensuring data quality, and maintaining comprehensive audit trails.

The architecture emphasizes decentralization, security by design, modularity, and extensibility. This initial design focuses on a robust, single-instance "reference node" that partners can deploy, with inherent extensibility to add scalability aspects as their needs grow.

4. High-Level Architecture

"Unlock effortless data exchange. The TSI DX Node consolidates your partner data exchanges into one simple, secure, and highly manageable solution"

A basic data flow diagram below



The Key Architectural Principles are

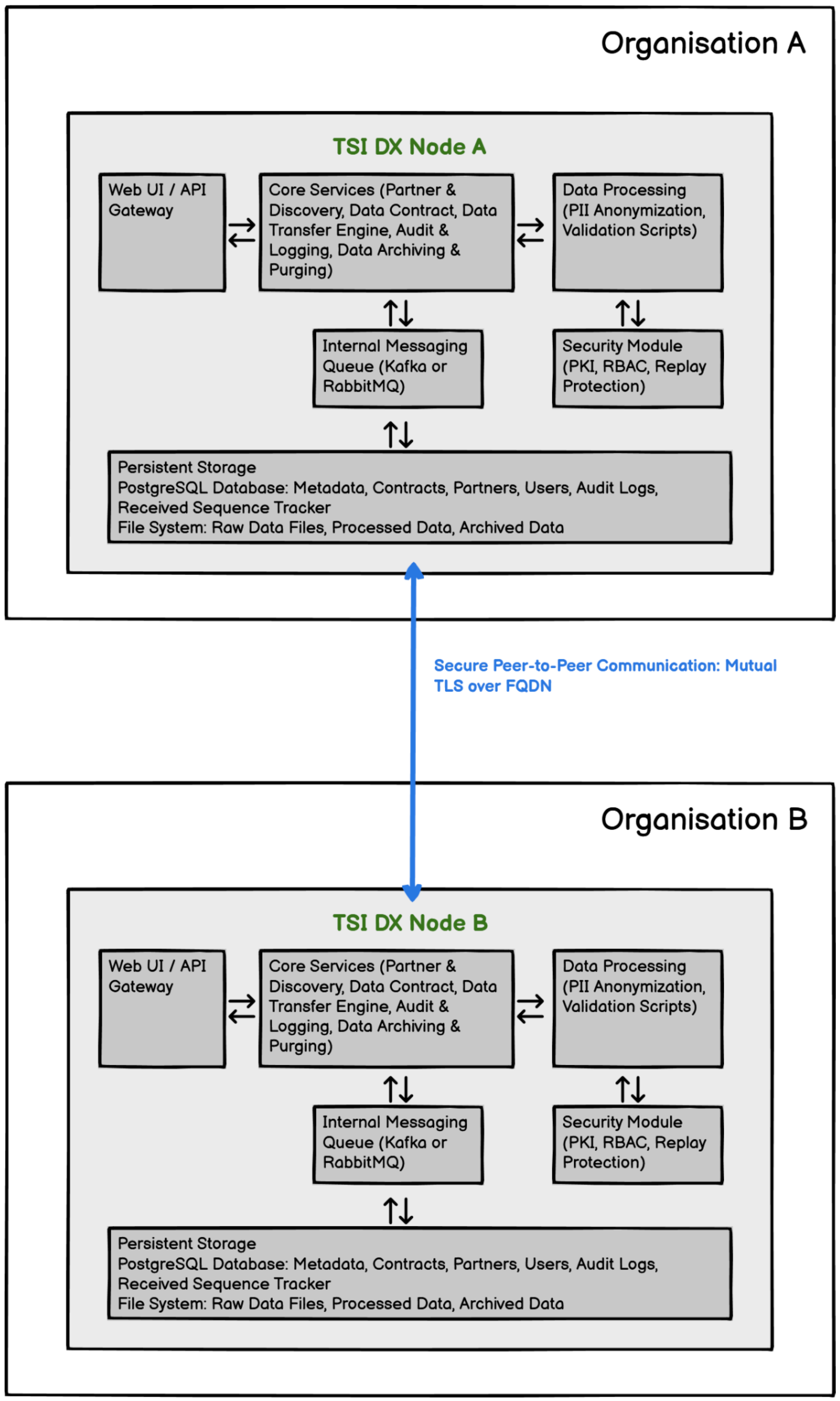
- **Decentralization:** No central authority or single point of control for data exchange. Each node operates autonomously while interacting with peers.
- **Security by Design:** Security is baked into every layer, from network communication (mTLS) to data processing (PII anonymization, replay protection) and access control (RBAC).
- **Modularity:** Components are loosely coupled, allowing for independent development, testing, and scaling.
- **API-First:** All functionalities are exposed via well-defined APIs, promoting programmatic access and integration.
- **Fault Tolerance:** Designed to withstand network interruptions and component failures, ensuring reliable data delivery.
- **Extensibility:** Built to allow for easy integration of new data formats, validation rules, and future features.

The TSI DX Node is conceived as a collection of interconnected services running within an organization's IT infrastructure. It acts as a secure gateway for incoming and outgoing data exchanges with registered partners. Key system design elements below.

- **TSI DX Protocol**: An Open-Source Standard for Decentralized Peer-to-Peer Data Exchange

- **Decentralized Network**: The system operates as a peer-to-peer network where each participating organization hosts its own TSI DX Node, implementing the TSI DX Protocol.
- **Organizational Autonomy**: Each organization (e.g., Organization A, Organization B) maintains its independent DX Node.
- **Core Node Components**: Within each TSI DX Node, the following main functional areas are interconnected:
  - **Web UI / API Gateway**: Provides interfaces for users and internal applications to interact with the node.
  - **Core Services**: Encompasses key functionalities like Partner & Discovery, Data Contract management, Data Transfer Engine, Audit & Logging, and Data Archiving & Purging.
  - **Data Processing**: Handles PII anonymization and data validation through scripting.
  - **Security Module**: Manages PKI, Role-Based Access Control (RBAC), and replay protection.
- **Internal Communication**: An Internal Message Queue (e.g., Kafka/RabbitMQ) facilitates asynchronous and fault-tolerant communication between the various Core Services and other modules within a single DX Node.
- **Persistent Storage**: Each node utilizes Persistent Storage (e.g., PostgreSQL for metadata, file system for data/archives, sequence tracker) to store its configurations, operational data, and audit trails.
- **Secure Peer-to-Peer Communication**: TSI DX Nodes communicate directly and securely with each other using mutual TLS (mTLS) over their Fully Qualified Domain Names (FQDNs), ensuring authenticated and encrypted data exchange.
- **Symmetric Architecture**: All participating TSI DX Nodes share a similar internal architecture, enabling standardized and interoperable data exchange across the network.

Conceptual Diagram below


Organisation A
TSI DX Node A
Web UI / API Gateway
Core Services (Partner & Discovery, Data Contract, Data Transfer Engine, Audit & Logging, Data Archiving & Purging)
Data Processing (PII Anonymization, Validation Scripts)
Internal Messaging Queue (Kafka or RabbitMQ)
Security Module (PKI, RBAC, Replay Protection)
Persistent Storage
PostgreSQL Database: Metadata, Contracts, Partners, Users, Audit Logs, Received Sequence Tracker
File System: Raw Data Files, Processed Data, Archived Data
Secure Peer-to-Peer Communication: Mutual TLS over FQDN
Organisation B
TSI DX Node B
Web UI / API Gateway
Core Services (Partner & Discovery, Data Contract, Data Transfer Engine, Audit & Logging, Data Archiving & Purging)
Data Processing (PII Anonymization, Validation Scripts)
Internal Messaging Queue (Kafka or RabbitMQ)
Security Module (PKI, RBAC, Replay Protection)
Persistent Storage
PostgreSQL Database: Metadata, Contracts, Partners, Users, Audit Logs, Received Sequence Tracker
File System: Raw Data Files, Processed Data, Archived Data

High-Level System Components below

1. **TSI DX Node Application:** The core software running on the organization's infrastructure.
2. **Internal Message Queue:** Facilitates asynchronous communication and fault tolerance within the node.
3. **Persistent Storage:** Stores metadata, configurations, audit logs, and temporary data.
4. **External Integrations:** APIs and UI for user/application interaction, and the secure P2P communication channel with other TSI DX Nodes.

5. TSI DX Protocol

The **TSI DX Protocol** is the set of rules, procedures, and data formats that govern how TSI DX Nodes interact to achieve secure, governed, and standardized peer-to-peer data exchange between business partners. It defines the "language" and "handshake" that nodes use to communicate reliably and trustworthily.

Based on the design of the TSI DX Node, the TSI DX Protocol encompasses the following key aspects:

1. **Node Identity and Discovery:**
    - **Unique Node Identification:** Each TSI DX Node is identified by a unique `node_id` and a globally resolvable `FQDN` (Fully Qualified Domain Name, typically a CNAME).
    - **Public Key Association:** Every node's identity is cryptographically bound to a `public_key_pem` and its `public_key_fingerprint`, which are shared during partner registration and discovery.
    - **Decentralized Discovery:** The protocol supports mechanisms for nodes to broadcast their presence and query known partners for their FQDNs and public keys, forming a decentralized partner registry.
2. **Secure Connection Establishment (Mutual TLS - mTLS):**
    - **Mandatory Mutual Authentication:** Before any data exchange, the protocol mandates a mutual TLS (mTLS) handshake between the sending and receiving TSI DX Nodes.
    - **Certificate Validation:** During mTLS, each node presents its X.509 certificate, which the other node validates against its known public key and verifies that the certificate's Common Name (CN) or Subject Alternative Names (SAN) match the expected FQDN of the connecting node.
    - **Encrypted Channel:** Upon successful mTLS, an encrypted, confidential communication channel (TLS 1.3) is established, ensuring data-in-transit confidentiality and integrity.
3. **Data Contract Negotiation and Enforcement:**

    - **Contract Definition:** The protocol defines how data contracts are structured, including schema definitions (e.g., JSON Schema, Avro), PII field declarations, data retention policies, and references to validation scripts.
    - **Lifecycle Management:** It specifies the states and transitions for contracts (e.g., `Draft` -> `Proposed` -> `Active` -> `Terminated`), requiring mutual agreement between involved partners for activation or amendment.
    - **Pre-transfer Validation:** Before actual data transmission, the sending node *must* apply the agreed-upon data contract rules, including schema validation and PII anonymization, as defined by the protocol.
4. **Data Transfer Mechanics (Message-Based):**
    - **Message Structure:** The protocol defines a standardized message format for data transfers, including a header and a payload.
    - **Mandatory Header Fields:** Each data transfer message header *must* include:
        - `sender_node_id`
        - `receiver_node_id`
        - `contract_id` (referencing the active data contract)
        - `sequence_number` (a monotonically increasing identifier for replay protection)
        - `message_timestamp` (UTC timestamp of message generation for freshness)
        - `message_hash` (cryptographic hash of the header and payload for integrity)
    - **Cryptographic Signing:** The entire message (header + payload) *must* be cryptographically signed by the sending node's private key.
    - **Chunking/Streaming:** The protocol supports efficient transfer of large data payloads through chunking or streaming mechanisms over the established mTLS channel.
    - **Broadcast Capability:** The protocol allows a single sender to initiate a transfer to multiple recipient nodes simultaneously, adhering to individual contracts with each.
5. **Replay Protection:**
    - **Sequence and Timestamp Validation:** The receiving node *must* validate incoming messages by checking their `sequence_number` and `message_timestamp` against the last successfully processed values from that specific sender and contract.
    - **Rejection of Invalid Messages:** Messages that are older or have a non-incrementing sequence number (indicating a replay attack) *must* be rejected at the protocol level.
    - **Atomic State Update:** Upon successful validation, the receiving node *must* atomically update its record of the `last_received_sequence_number` and `last_received_timestamp` for that sender-contract pair.
6. **Acknowledgement and Status Reporting:**

    - **Transfer Confirmations:** The protocol defines mechanisms for receiving nodes to send acknowledgments (success/failure) back to the sending node upon completion of data reception and initial processing.
    - **Error Reporting:** Standardized error codes and messages are part of the protocol to facilitate debugging and fault resolution.
7. **Auditability:**
    - The protocol implicitly requires that all significant events related to node interaction and data exchange (e.g., connection attempts, transfer initiation, receipt, validation results, security events) are logged by participating nodes, contributing to a decentralized audit trail.

In essence, the TSI DX Protocol is the blueprint that ensures all TSI DX Nodes can speak the same secure, reliable, and governed language for inter-organizational data exchange.

6. Component Overview

3.1. Web UI / API Gateway

- **Responsibilities:**
    - Provides a user-friendly web interface for administrators and authorized users to manage the TSI DX Node.
    - Exposes RESTful API endpoints for programmatic interaction by internal applications.
    - Acts as an API Gateway, handling authentication (API keys, JWT), request routing, and basic input validation before forwarding to internal services.
- **Technologies:**
    - **Web UI:** React/Vue.js/Angular (frontend), consuming the API Gateway.
    - **API Gateway:** Go (e.g., Gin/Echo framework) or Node.js (e.g., Express).
- **Interactions:** Communicates with all Core Services via internal APIs.

3.2. Core Services

This is the heart of the TSI DX Node, comprising several interconnected microservices or modules.

3.2.1. Partner & Discovery Service

- **Responsibilities:**
    - Manages the registry of the local TSI DX Node and all known partner TSI DX Nodes.
    - Handles partner registration (manual and self-discovery).
    - Performs DNS lookups for partner FQDNs.
    - Manages the `ReceivedSequenceTracker` for replay protection.
- **Data Models:** `Partner`, `ReceivedSequenceTracker`.

- **Interactions:**
    - Communicates with other TSI DX Nodes for auto-discovery (via P2P Communication Module).
    - Persists data to the Metadata Database.
    - Consulted by the Transfer Engine for partner details.

3.2.2. Data Contract Service

- **Responsibilities:**
    - Stores and manages data contract definitions (schema, PII rules, retention, validation script references).
    - Manages the lifecycle of contracts (draft, proposed, active, terminated).
    - Provides contract enforcement rules to the Data Processing module.
- **Data Models:** `DataContract`, `ValidationScript`.
- **Interactions:**
    - Persists data to the Metadata Database.
    - Consulted by the Data Processing module for contract rules.

3.2.3. Data Transfer Engine

- **Responsibilities:**
    - Orchestrates the end-to-end data transfer process (upload, processing, sending, receiving, downloading).
    - Manages transfer queues and retries (using Internal Message Queue).
    - Tracks the status and progress of all transfers.
    - Generates a new, unique `sequence_number` and `timestamp` for *every initiated transfer*, including legitimate re-sends, to ensure each transfer event is distinct for replay protection.
    - Validates `sequence_number` and `timestamp` for incoming messages.
- **Data Models:** `DataTransfer`.
- **Interactions:**
    - Publishes/consumes messages from the Internal Message Queue.
    - Interacts with Partner Service to get partner FQDNs.
    - Invokes Data Processing for pre-send/post-receive transformations.
    - Utilizes Security Module for cryptographic operations.
    - Communicates with other TSI DX Nodes via the P2P Communication Module.
    - Persists data to the Metadata Database.

3.2.4. Audit & Logging Service

- **Responsibilities:**
    - Captures and stores all significant events within the TSI DX Node.
    - Ensures log integrity (tamper-proofing).
    - Provides search, filtering, and export capabilities for audit logs.
    - Supports forwarding logs to external SIEM systems.

- **Data Models:** `AuditLog`.
- **Interactions:**
    - Receives events from all other services.
    - Persists data to the Metadata Database.

3.2.5. Data Archiving & Purging Service

- **Responsibilities:**
    - Implements data retention policies based on contract definitions.
    - Automatically moves data from active storage to archive.
    - Securely purges data after its defined retention period.
    - Records all archiving and purging events in the Audit Log.
- **Interactions:**
    - Accesses the File System Storage for data files.
    - Consults Data Contract Service for retention rules.
    - Sends events to Audit & Logging Service.

3.3. Data Processing Module

- **Responsibilities:**
    - Performs PII anonymization/pseudonymization based on contract rules.
    - Executes custom data validation scripts (Python/JavaScript).
    - Handles data format transformations (if required by contract).
- **Technologies:**
    - **PII Anonymization:** Custom implementation of hashing, masking, tokenization algorithms.
    - **Scripting Engine:** Embedded Python interpreter (e.g., CPython bindings) or JavaScript runtime (e.g., V8).
    - **Sandboxing:** OS-level containerization (Docker, namespaces) or language-specific sandboxing for secure script execution.
- **Interactions:**
    - Receives data and contract rules from the Data Transfer Engine.
    - Returns processed data or validation results.
    - Sends events to Audit & Logging Service.

3.4. Security Module

- **Responsibilities:**
    - Manages PKI (certificate generation, import, validation, revocation checks).
    - Handles secure key storage and usage.
    - Enforces Role-Based Access Control (RBAC) for local users and API access.
    - Performs cryptographic operations (signing, encryption, decryption).
    - Implements replay protection mechanisms (sequence number/timestamp validation) to prevent unauthorized re-transmissions. **Crucially, it differentiates**

    between malicious replay attempts and legitimate re-sends initiated by authorized parties.
- **Technologies:**
    - **PKI:** Go's `crypto/tls`, `crypto/x509` (or similar language-specific libraries), OpenSSL integration.
    - **Key Storage:** File system encryption, potentially integration with KMS/HSM.
    - **RBAC:** Internal authorization logic based on user roles and permissions stored in the Metadata Database.
- **Interactions:**
    - Consulted by API Gateway for user authentication/authorization.
    - Used by Partner Service for certificate management during discovery.
    - Used by Data Transfer Engine for mTLS, message signing/verification, and replay protection checks.

3.5. Internal Message Queue

- **Responsibilities:**
    - Provides asynchronous communication between internal TSI DX Nodes services.
    - Ensures message durability and guaranteed delivery (retries, dead-letter queues).
    - Buffers requests to handle spikes in load.
- **Technologies:** Apache Kafka or RabbitMQ.
- **Interactions:** All core services publish and subscribe to relevant topics/queues.

3.6. Persistent Storage

- **Responsibilities:**
    - **Metadata Database:** Stores all structured metadata (Partners, Contracts, Transfers, Audit Logs, Users, Roles, Sequence Tracker).
    - **File System Storage:** Stores actual data files during transfer, pre-processed data, and archived data.
- **Technologies:**
    - **Metadata DB:** PostgreSQL (primary choice for robustness and features), or SQLite (for simpler deployments).
    - **File System:** Local disk, Network File System (NFS), or cloud object storage (e.g., S3-compatible) for archiving.
- **Interactions:** All services that manage state or data interact with Persistent Storage.

7. Data Flow: End-to-End Data Transfer (Sender to Receiver)

This section illustrates a typical data transfer lifecycle, incorporating replay protection.

Scenario: Organization A sends data to Organization B under an active data contract.

1. **Initiation (Org A - Sender):**
   - User/Application uploads data via Web UI or API Gateway.
   - API Gateway authenticates user/app and authorizes the request based on RBAC.
   - Request is sent to Data Transfer Engine.
2. **Pre-Transfer Processing (Org A - Sender):**
   - Data Transfer Engine retrieves `DataContract` details from Data Contract Service.
   - **For every initiated transfer (including re-sends for genuine reasons), the Data Transfer Engine generates a new, unique `sequence_number` and a fresh `timestamp` for this specific transfer to Org B under this contract.** This ensures that each transfer event, even if carrying identical data, is treated as a distinct, valid message by the replay protection mechanism. The `ReceivedSequenceTracker` (for its *own* outgoing sequence) is updated.
   - Data is passed to Data Processing Module.
   - Data Processing Module applies PII anonymization and runs pre-transfer validation scripts based on contract rules. If validation fails, transfer is aborted, and Audit Log is updated.
   - Data Transfer Engine computes a `message_hash` of the prepared data and header.
3. **Secure Transmission (Org A to Org B):**
   - Data Transfer Engine initiates an outbound connection to TSI DX Node B's FQDN.
   - **mTLS Handshake:** Security Module performs mutual TLS handshake with TSI DX Node B, validating certificates (CN/SAN matches FQDN) and establishing an encrypted channel.
   - Data Transfer Engine constructs the message, including `sender_node_id`, `receiver_node_id`, `contract_id`, `sequence_number`, `timestamp`, `message_hash`, and the processed data.
   - Security Module cryptographically signs the entire message using TSI DX Node A's private key.
   - Signed, encrypted data is streamed over the mTLS channel to TSI DX Node B.
4. **Reception & Validation (Org B - Receiver):**
   - TSI DX Node B's P2P Communication Module receives the incoming connection.
   - **mTLS Handshake:** TSI DX Node B's Security Module performs mutual TLS handshake, authenticating TSI DX Node A.
   - Incoming message is passed to Data Transfer Engine.
   - **Signature Verification:** Data Transfer Engine requests Security Module to verify the message's cryptographic signature using TSI DX Node A's known public key. If invalid, reject.

    - **Replay Protection Check:** Data Transfer Engine retrieves `last_received_sequence_number` and `last_received_timestamp` for TSI DX Node A and the specific contract from its `ReceivedSequenceTracker`.
        - If `incoming_sequence_number <= last_received_sequence_number` or `incoming_timestamp` is significantly older than the last successfully processed message (beyond a configurable freshness window), the message is rejected as a replay **attack**.
        - If checks pass, `ReceivedSequenceTracker` is atomically updated with the new `sequence_number` and `timestamp`.
    - Data is passed to Data Processing Module.
    - Data Processing Module runs post-reception validation scripts. If validation fails, data is quarantined/rejected, and Audit Log is updated.
5. **Delivery & Confirmation (Org B - Receiver):**
    - If all validations pass, data is delivered to the configured internal destination (e.g., local directory, internal application).
    - TSI DX Node B sends a successful acknowledgment back to TSI DX Node A via the secure channel.
    - Audit Logs are updated on both nodes.
6. **Post-Transfer (Both Nodes):**
    - Data Archiving & Purging Service on both nodes applies retention policies to the transferred data and associated metadata.

4.1. Handling Legitimate Re-sends vs. Replay Attacks

The TSI DX Node differentiates between a malicious replay attack and a legitimate re-send for genuine business reasons:

- **Malicious Replay Attack:** An unauthorized re-transmission of a *previously sent and processed message* by an attacker. These attempts will be blocked by the replay protection mechanism because the `incoming_sequence_number` will be less than or equal to the `last_received_sequence_number`, or the `incoming_timestamp` will be significantly older than expected.
- **Legitimate Re-send:** An explicit action initiated by an *authorized user or system* on the sending TSI DX Node. When a re-send is performed, the sending TSI DX Node **generates a brand new `sequence_number` and a fresh `timestamp`** for that specific transfer event. This makes the re-sent message appear as a new, valid transaction to the receiving TSI DX Node, allowing it to pass the replay protection checks and be processed. This ensures flexibility for scenarios like error correction, re-transmission of missing data, or system recovery.

8. Admin App

The Admin App serves as the primary interface for managing and monitoring the TSI DX Node. Its design prioritizes clarity, ease of use, and efficient access to critical functionalities.

1. Overall Layout (Common to all pages):

- **Purpose:** To provide a consistent and intuitive navigation experience across the entire application, ensuring users can easily find and access different features.
- **Elements & Rationale:**
    - **Header (Top Bar):** Contains persistent elements like the application logo/name for branding, the Node ID for immediate identification of the active node, and user-specific controls (profile, settings, logout) for account management. An optional global search bar allows quick access to any entity within the system.
    - **Sidebar Navigation (Left Panel):** This is the primary navigation hub. Being collapsible saves screen real estate on smaller displays while providing quick access to major sections. Grouping related functionalities (e.g., `Configuration` with sub-menus) improves organization and reduces clutter.
    - **Main Content Area (Right/Center):** This dynamic area is where the core work happens. Breadcrumbs provide context, showing the user their current location within the application's hierarchy, which is especially useful in multi-level sections. The Page Title clearly identifies the current view.

2. Dashboard (Landing Page):

- **Purpose:** To offer an at-a-glance summary of the DX Node's operational status, key metrics, and immediate attention items, allowing administrators to quickly assess system health.
- **Elements & Rationale:**
    - **Layout:** A grid or card-based layout is chosen for its ability to present multiple pieces of information concisely and visually. Each card can represent a distinct operational area.
    - **Key Sections/Cards:** These cards highlight the most critical information:
        - **Node Status:** Essential for confirming the node is running correctly and its network identity.
        - **Active/Recent Transfers:** Gives immediate insight into ongoing and recently completed work, showing throughput.
        - **Pending Actions:** Acts as a "to-do" list, drawing attention to items requiring administrator intervention (e.g., contract approvals, failed transfers).
        - **Storage Usage:** Provides a quick check on resource consumption, preventing potential issues due to full disks.
        - **Recent Audit Events:** Offers a snapshot of recent system activities, particularly useful for spotting anomalies or critical security events.
    - **Quick Links:** Many cards include quick links to detailed lists, enabling users to drill down for more information without navigating through menus.

3. Partners Section:

- **Purpose:** To manage the registry of all known DX Node partners, enabling administrators to add, view, edit, and remove partner configurations.
- **Elements & Rationale:**
    - **Layout:** A data table is ideal for displaying lists of entities, allowing for easy scanning and comparison.
    - **Top Area:**
        - `Page Title` and `+ Add New Partner` button are standard for list views, providing clear context and the primary action.
        - `Search Input` and `Filter Dropdowns` are crucial for managing potentially large lists of partners, allowing users to quickly find specific entries or narrow down results.
    - **Partner List Table:** Columns are chosen to display essential partner identification and status information. The `Actions` column provides direct access to common operations for each partner, improving efficiency.
    - **Add/Edit Partner Form:** This form (modal or separate page) collects all necessary details for a partner. Using a textarea/upload for `Public Key` facilitates both direct input and file-based import.

4. Data Contracts Section:

- **Purpose:** To define, manage the lifecycle of, and enforce data sharing agreements between the local node and its partners.
- **Elements & Rationale:**
    - **Layout:** Similar to Partners, a data table is used for listing contracts.
    - **Top Area:** Standard `Page Title`, `+ Create New Contract` button, `Search Input`, and `Filter Dropdowns` (e.g., by status or direction) for efficient management.
    - **Contract List Table:** Columns display key contract attributes. The `Actions` column includes contextual buttons (e.g., `Propose/Accept/Terminate`) that change based on the contract's current status, guiding the user to valid next steps.
    - **Create/Edit Contract Form:** A multi-step wizard or tabbed form is used because data contracts are complex and involve multiple distinct sets of information (general info, schema, governance rules). This breaks down complexity into manageable steps, improving user experience. `Save Draft` allows for incomplete work, and `Propose` triggers the workflow.

5. Data Transfers Section:

- **Purpose:** To allow users to initiate new data transfers, monitor their progress, and review the history of all incoming and outgoing data exchanges.
- **Elements & Rationale:**

        - **Layout:** A tabbed interface (`Active`, `Completed`, `Failed`) is used to logically separate transfers by their current state, making it easy for users to focus on what's relevant. Each tab contains a data table.
        - **Common Elements per Tab:** `+ Initiate New Transfer` button (on the `Active` tab) is the primary action. `Search Input` and `Filter Dropdowns` enable efficient searching through transfer records.
        - **Transfer List Table:** Displays key information about each transfer. The `Actions` column includes `Download` for received files and `Re-send` for failed or completed transfers, providing direct operational capabilities.
        - **Initiate New Transfer Form:** This form collects all necessary details for a transfer, including multi-select for partners (for broadcast) and file upload options. `Upload & Send` button clearly indicates the action.

6. Users & Roles Section:

- **Purpose:** To manage user accounts and define access permissions within the Admin App, ensuring proper Role-Based Access Control (RBAC).
- **Elements & Rationale:**
    - **Layout:** Divided into two main sections (Users and Roles) to separate user account management from permission definition.
    - **Users Tab/Section:** A table lists users with actions for editing, password resets, and activation/deactivation. `+ Add New User` is the primary creation action.
    - **Roles Tab/Section:** A table lists roles with actions for editing permissions and deleting custom roles. `+ Create New Role` allows for flexible permission management.
    - **Edit User/Role Forms:** These forms provide the necessary fields for managing user credentials and assigning roles, or for defining granular permissions for roles using a clear checkbox matrix, which visually represents allowed actions on different resources.

7. Audit Logs Section:

- **Purpose:** To provide a comprehensive, searchable, and exportable record of all activities within the DX Node, crucial for security, compliance, and troubleshooting.
- **Elements & Rationale:**
    - **Layout:** A data table is used to display log entries.
    - **Top Area:** `Page Title` and `Export Logs` button (for compliance/external analysis). Extensive `Filter Dropdowns` (Event Type, User, Entity ID, Date/Time, Severity) are critical for navigating potentially massive log volumes.
    - **Audit Log Table:** Columns display key log attributes. `Details (truncated)` provides a summary, with a `View Details` button to show the full JSON payload of a log entry in a modal, allowing for deep inspection.

8. Configuration Section (Sub-menu items):

- **Purpose:** To allow administrators to configure the fundamental operational parameters, security settings, and governance rules of the DX Node.
- **Elements & Rationale:**
  - **Sub-menu Items:** Grouping configuration settings into logical sub-sections (Node, PII, Scripts, Archiving/Purging) improves navigability and prevents a single, overwhelming configuration page.
  - **Node Settings:** Contains core operational settings and PKI management actions (CSR generation, certificate import), which are fundamental to the node's identity and communication.
  - **PII Anonymization Rules:** Provides a dedicated interface to define and manage how sensitive data fields are handled, directly supporting data governance requirements.
  - **Validation Scripts:** Allows for the upload and management of custom code that enhances data quality and contract enforcement. A code editor would be ideal here.
  - **Archiving & Purging Policies:** Enables setting data retention and deletion rules, critical for compliance and storage management.
  - **Save Changes Button:** Standard for configuration forms, ensuring changes are explicitly applied.

9. API Design

This section details the RESTful API endpoints for interacting with the TSI DX Node. It is divided into two main sections: the Admin API, used for managing the node's configuration and operations, and the Client API, used by internal applications to initiate and monitor data exchanges.

1. General API Principles

- **RESTful Design:** Use standard HTTP methods (GET, POST, PUT, DELETE) and resource-oriented URLs.
- **JSON Payloads:** All request and response bodies will be in JSON format.
- **Authentication:** All API endpoints require authentication.
- **Authorization:** Role-Based Access Control (RBAC) will govern access to specific endpoints and actions.
- **Versioning:** APIs will be versioned (e.g., `/api/v1/`).
- **Error Handling:** Consistent error response structure with meaningful error codes and messages.
- **Pagination & Filtering:** Support for pagination, filtering, sorting, and searching on list endpoints.

1.1. Common Response Structure

- **Success:**
  JSON

```
None

{
  "success": true,
  "data": { /* Resource object or array of objects */ },
  "message": "Operation successful." // Optional
}
```

-
- **Error:**
  JSON

```
None

{
  "success": false,
  "error": {
    "code": "ERROR_CODE_ENUM", // e.g., "VALIDATION_ERROR",
"NOT_FOUND", "UNAUTHORIZED"
    "message": "A human-readable error message.",
    "details": { /* Optional: specific validation errors, field
names, etc. */ }
  }
}
```

-

1.2. Authentication

All API requests must include appropriate authentication credentials.

- **Admin UI/User Access (for Admin API):** Uses JWT (JSON Web Tokens) obtained after a successful login. `Authorization: Bearer <JWT_TOKEN>`
- **Client Applications (for Client API):** Uses API Keys and API Secrets for programmatic, machine-to-machine access. `X-API-Key: <YOUR_API_KEY>` `X-API-Secret:`

`<YOUR_API_SECRET>` (API Keys and Secrets would be managed by an Admin user via the Admin UI/API).

---

2. Admin API

Base URL: `/api/v1/admin` **Authentication:** `Authorization: Bearer <JWT_TOKEN>` **Authorization:** Governed by RBAC roles (e.g., Administrator, Auditor).

2.1. Node Management

- **GET `/status`**
  - **Description:** Get the overall status and health of the DX Node.
  - **Response:** `200 OK`
    JSON

```
None

{
  "success": true,
  "data": {
    "node_id": "dx-node-org-a-123",
    "fqdn": "dxnode.yourcompany.com",
    "status": "Online", // "Online", "Degraded", "Offline"
    "last_heartbeat": "2025-07-16T15:30:00Z",
    "uptime_seconds": 3600,
    "disk_usage_gb": { "total": 500, "used": 150, "archive": 50
},
    "active_transfers_count": 5,
    "pending_transfers_count": 2,
    "failed_transfers_count": 1
  }
}
```

  -
- **GET `/config`**
  - **Description:** Retrieve the current node configuration.

    - **Response:** `200 OK` (sensitive info like private keys excluded)
- PUT `/config`
    - **Description:** Update node configuration (e.g., network ports, storage paths, logging levels).
    - **Request Body:** JSON object with configurable fields.
    - **Response:** `200 OK`
- POST `/config/pki/csr`
    - **Description:** Generate a Certificate Signing Request (CSR) for the node's FQDN.
    - **Request Body:** `{"common_name": "dxnode.yourcompany.com", "organization": "Your Org", ...}`
    - **Response:** `200 OK` with CSR in PEM format.
- POST `/config/pki/certificate`
    - **Description:** Import a signed certificate and private key.
    - **Request Body:** `{"certificate_pem": "...", "private_key_pem": "..."}`
    - **Response:** `200 OK`

2.2. Partner Management

- GET `/partners`
    - **Description:** List all registered partners.
    - **Query Params:** `status`, `search`, `page`, `limit`.
    - **Response:** `200 OK` with array of partner objects.
      JSON

```
None

{
  "success": true,
  "data": [
    {
      "partner_id": "uuid-partner-1",
      "node_id": "partner-node-alpha",
      "name": "Alpha Corp",
      "fqdn": "dx.alphacorp.com",
      "status": "Active", // "Active", "Pending", "Inactive"
      "public_key_fingerprint": "SHA256:...", // Fingerprint for
display
```

```
      "last_active": "2025-07-16T15:25:00Z"
    }
  ],
  "pagination": { "total": 10, "page": 1, "limit": 10 }
}
```

  ○

- **POST `/partners`**
  - **Description:** Register a new partner.
  - **Request Body:** `{"name": "...", "node_id": "...", "fqdn": "...", "public_key_pem": "..."}`
  - **Response:** `201 Created` with new partner object.
- **GET `/partners/{partner_id}`**
  - **Description:** Get details of a specific partner.
  - **Response:** `200 OK` with partner object.
- **PUT `/partners/{partner_id}`**
  - **Description:** Update a partner's details (e.g., FQDN, public key).
  - **Request Body:** Updated partner fields.
  - **Response:** `200 OK`
- **DELETE `/partners/{partner_id}`**
  - **Description:** Remove a partner.
  - **Response:** `204 No Content`

2.3. Data Contract Management

- **GET `/contracts`**
  - **Description:** List all data contracts.
  - **Query Params:** `status`, `direction`, `partner_id`, `search`, `page`, `limit`.
  - **Response:** `200 OK` with array of contract objects.
- **POST `/contracts`**
  - **Description:** Create a new data contract.
  - **Request Body:** `{"name": "...", "version": 1, "sender_partner_id": "...", "receiver_partner_id": "...", "schema_definition": {}, "metadata": {}, "validation_script_id": "..."}`
  - **Response:** `201 Created` with new contract object.
- **GET `/contracts/{contract_id}`**
  - **Description:** Get details of a specific data contract.

    - **Response:** `200 OK` with contract object.
- PUT `/contracts/{contract_id}`
    - **Description:** Update an existing data contract.
    - **Request Body:** Updated contract fields.
    - **Response:** `200 OK`
- POST `/contracts/{contract_id}/propose`
    - **Description:** Propose a contract to a partner.
    - **Response:** `200 OK`
- POST `/contracts/{contract_id}/accept`
    - **Description:** Accept a proposed contract (by receiver).
    - **Response:** `200 OK`
- POST `/contracts/{contract_id}/reject`
    - **Description:** Reject a proposed contract.
    - **Response:** `200 OK`
- POST `/contracts/{contract_id}/terminate`
    - **Description:** Terminate an active contract.
    - **Response:** `200 OK`

2.4. User & Role Management

- GET `/users`
    - **Description:** List all local users.
    - **Response:** `200 OK` with array of user objects.
- POST `/users`
    - **Description:** Create a new local user.
    - **Request Body:** `{"username": "...", "email": "...", "password": "...", "role_ids": ["uuid-role-1"]}`
    - **Response:** `201 Created` with new user object.
- GET `/users/{user_id}`
    - **Description:** Get details of a specific user.
    - **Response:** `200 OK`
- PUT `/users/{user_id}`
    - **Description:** Update a user's details or roles.
    - **Request Body:** Updated user fields.
    - **Response:** `200 OK`
- DELETE `/users/{user_id}`
    - **Description:** Delete a user.
    - **Response:** `204 No Content`
- GET `/roles`
    - **Description:** List all defined roles.
    - **Response:** `200 OK` with array of role objects.

- **POST `/roles`**
    - **Description:** Create a new custom role.
    - **Request Body:** `{"name": "...", "description": "...", "permissions": ["partner:view", "transfer:initiate"]}`
    - **Response:** `201 Created`
- **PUT `/roles/{role_id}`**
    - **Description:** Update a role's permissions.
    - **Request Body:** Updated role fields.
    - **Response:** `200 OK`
- **DELETE `/roles/{role_id}`**
    - **Description:** Delete a custom role.
    - **Response:** `204 No Content`

2.5. Data Transfer Monitoring & Management

- **GET `/transfers`**
    - **Description:** List all data transfers (active, completed, failed).
    - **Query Params:** `status`, `direction`, `partner_id`, `contract_id`, `start_date`, `end_date`, `search`, `page`, `limit`.
    - **Response:** `200 OK` with array of transfer objects.
- **GET `/transfers/{transfer_id}`**
    - **Description:** Get details of a specific data transfer.
    - **Response:** `200 OK` with transfer object (including status, audit trail references).
- **POST `/transfers/{transfer_id}/resend`**
    - **Description:** Initiate a legitimate re-send of a previously failed or completed transfer. This creates a *new* transfer event with a new sequence number.
    - **Response:** `202 Accepted` with a reference to the new transfer ID.

2.6. Audit Logs

- **GET `/auditlogs`**
    - **Description:** Retrieve audit log entries.
    - **Query Params:** `event_type`, `user_id`, `entity_type`, `entity_id`, `severity`, `start_time`, `end_time`, `search`, `page`, `limit`.
    - **Response:** `200 OK` with array of audit log objects.
- **GET `/auditlogs/{log_id}`**
    - **Description:** Get full details of a specific audit log entry.
    - **Response:** `200 OK`

2.7. Data Governance Configuration

- **GET `/governance/pii-rules`**

    - **Description:** List configured PII anonymization rules.
    - **Response:** `200 OK`
- POST `/governance/pii-rules`
    - **Description:** Add a new PII anonymization rule.
    - **Request Body:** `{"field_name": "...", "method": "HASH", "config": {}}`
    - **Response:** `201 Created`
- PUT `/governance/pii-rules/{rule_id}`
    - **Description:** Update an existing PII anonymization rule.
    - **Response:** `200 OK`
- DELETE `/governance/pii-rules/{rule_id}`
    - **Description:** Delete a PII anonymization rule.
    - **Response:** `204 No Content`
- GET `/governance/validation-scripts`
    - **Description:** List uploaded validation scripts.
    - **Response:** `200 OK`
- POST `/governance/validation-scripts`
    - **Description:** Upload a new validation script.
    - **Request Body:** `{"name": "...", "language": "PYTHON", "content": "print('hello')", "contract_ids": []}`
    - **Response:** `201 Created`
- PUT `/governance/validation-scripts/{script_id}`
    - **Description:** Update a validation script.
    - **Response:** `200 OK`
- DELETE `/governance/validation-scripts/{script_id}`
    - **Description:** Delete a validation script.
    - **Response:** `204 No Content`
- GET `/governance/archiving-purging`
    - **Description:** Get archiving and purging policies.
    - **Response:** `200 OK`
- PUT `/governance/archiving-purging`
    - **Description:** Update archiving and purging policies.
    - **Request Body:** `{"active_retention_days": 30, "archive_retention_days": 365, "archive_location": "/path/to/archive"}`
    - **Response:** `200 OK`

3. Client API

Base URL: `/api/v1/client` **Authentication:** `X-API-Key: <YOUR_API_KEY>` and `X-API-Secret: <YOUR_API_SECRET>` **Authorization:** Governed by RBAC roles (e.g., Data Uploader, Data Downloader).

3.1. Data Transfer Operations

- **POST `/transfers/initiate`**
  - **Description:** Initiate a new data transfer to one or more partners under a specific contract.
  - **Request Body:**
    JSON

```
None

{
  "target_partner_ids": ["uuid-partner-1", "uuid-partner-2"], //
Array for broadcast
  "contract_id": "uuid-contract-abc",
  "file_name": "sales_report_q2.csv",
  "file_content_base64": "...", // Base64 encoded file content
for smaller files
  "file_url": "sftp://...", // Or a URL for the DX Node to fetch
for larger files
  "metadata": { "report_date": "2025-06-30" } // Optional custom
metadata
}
```

  - *Note: For very large files, a separate mechanism like pre-signed URLs or direct file upload to a staging area might be considered, with this API then triggering the processing.*
  - **Response:** `202 Accepted`
    JSON

```
None

{
  "success": true,
  "data": {
    "transfer_id": "uuid-transfer-xyz",
    "status": "Pending",
    "message": "Transfer initiated successfully."
  }
}
```

  -
- GET `/transfers/{transfer_id}/status`
  - **Description:** Get the current status of a specific data transfer.
  - **Response:** `200 OK`
    JSON

```
None

{
  "success": true,
  "data": {
    "transfer_id": "uuid-transfer-xyz",
    "status": "Processing", // "Pending", "Processing", "Sent",
"Received", "Failed", "Delivered"
    "progress_percentage": 50, // For ongoing transfers
    "last_update_time": "2025-07-16T15:35:00Z",
    "error_message": null // If failed
  }
}
```

  -
- GET `/transfers/received`
  - **Description:** List data transfers received by this node, ready for download/processing.
  - **Query Params:** `status` (e.g., `READY_FOR_DOWNLOAD`, `PROCESSED`), `sender_partner_id`, `contract_id`, `start_date`, `end_date`, `page`, `limit`.
  - **Response:** `200 OK` with array of received transfer objects.
- GET `/transfers/received/{transfer_id}/download`

- **Description:** Download the content of a specific received data transfer.
- **Response:** `200 OK` with file content (e.g., `Content-Type: application/octet-stream`). *Note: This might involve streaming the file directly or providing a temporary download URL.*

10. Database Schema

1. `partners` Table

Stores information about registered DX Node partners.

Column Name	Data Type	Constraints	Description
`partner_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the partner.
`node_id`	`VARCHAR(255)`	`NOT NULL`, `UNIQUE`	The unique identifier of the partner's DX Node.
`name`	`VARCHAR(255)`	`NOT NULL`	Human-readable name of the partner organization.
`fqdn`	`VARCHAR(255)`	`NOT NULL`, `UNIQUE`	Fully Qualified Domain Name of the partner's node (CNAME).
`public_key_pem`	`TEXT`	`NOT NULL`	PEM-encoded public key of the partner's node.

public_key_fingerprint	VARCHAR(255)	NOT NULL, UNIQUE	Cryptographic fingerprint of the public key (e.g., SHA256).
status	VARCHAR(50)	NOT NULL, DEFAULT 'Pending'	Current status: 'Pending', 'Active', 'Inactive'.
created_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Timestamp when the partner record was created.
updated_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Timestamp when the partner record was last updated.

2. `users` Table

`users` Table (MODIFIED)

Now includes a direct `role_id` foreign key, ensuring each user has exactly one role.

Column Name	Data Type	Constraints	Description
user_id	UUID	PRIMARY KEY, NOT NULL, DEFAULT gen_random_uuid()	Unique identifier for the user.
username	VARCHAR(100)	NOT NULL, UNIQUE	User's login username.

`email`	`VARCHAR(255)`	`NOT NULL`, `UNIQUE`	User's email address.
`password_hash`	`VARCHAR(255)`	`NOT NULL`	Hashed password for security.
`role_id`	`UUID`	`NOT NULL`, `FOREIGN KEY REFERENCES roles(role_id)`	**Foreign key to the `roles` table, defining the user's single role.**
`status`	`VARCHAR(50)`	`NOT NULL`, `DEFAULT 'Active'`	User status: 'Active', 'Inactive', 'Locked'.
`last_login_at`	`TIMESTAMP WITH TIME ZONE`	`NULL`	Timestamp of the user's last successful login.
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the user record was created.
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the user record was last updated.

3. `roles` Table

Defines different roles within the Admin App.

Column Name	Data Type	Constraints	Description

`role_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the role.
`name`	`VARCHAR(100)`	`NOT NULL`, `UNIQUE`	Name of the role (e.g., 'Administrator', 'DataUploader').
`description`	`TEXT`	`NULL`	Description of the role's responsibilities.
`is_system_role`	`BOOLEAN`	`NOT NULL`, `DEFAULT FALSE`	True for pre-defined system roles, False for custom roles.
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the role record was created.
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the role record was last updated.

4. `role_permissions` Table

Stores granular permissions for each role.

Column Name	Data Type	Constraints	Description

`permission_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the permission.
`role_id`	`UUID`	`NOT NULL`, `FOREIGN KEY REFERENCES roles(role_id)`	Foreign key to the `roles` table.
`resource`	`VARCHAR(100)`	`NOT NULL`	The resource being accessed (e.g., 'partner', 'transfer', 'config').
`action`	`VARCHAR(100)`	`NOT NULL`	The action allowed on the resource (e.g., 'view', 'create', 'update', 'delete', 'initiate', 'download').
		`UNIQUE (role_id, resource, action)`	Ensures unique permission per role.

5. `api_keys` Table

Stores API keys and secrets for Client API access.

Column Name	Data Type	Constraints	Description
`key_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the API key.

api_key	VARCHAR(255)	NOT NULL, UNIQUE	The public API key.
api_secret_hash	VARCHAR(255)	NOT NULL	Hashed API secret for security.
user_id	UUID	NULL, FOREIGN KEY REFERENCES users(user_id)	Optional: User who generated/owns this key.
description	TEXT	NULL	Description of the API key's purpose.
status	VARCHAR(50)	NOT NULL, DEFAULT 'Active'	Key status: 'Active', 'Inactive', 'Revoked'.
created_at	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Timestamp when the key was created.
expires_at	TIMESTAMP WITH TIME ZONE	NULL	Optional: Expiration timestamp for the key.
last_used_at	TIMESTAMP WITH TIME ZONE	NULL	Timestamp of the last successful API call using this key.

6. `validation_scripts` Table

Stores custom data validation scripts.

Column Name	Data Type	Constraints	Description
`script_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the script.
`name`	`VARCHAR(255)`	`NOT NULL`, `UNIQUE`	Name of the validation script.
`language`	`VARCHAR(50)`	`NOT NULL`	Scripting language: 'PYTHON', 'JAVASCRIPT'.
`content`	`TEXT`	`NOT NULL`	The actual script code.
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the script was uploaded.
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the script was last updated.

7. `data_contracts` Table

Defines the agreements for data exchange between partners.

Column Name	Data Type	Constraints	Description

`contract_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the data contract.
`name`	`VARCHAR(255)`	`NOT NULL`	Human-readable name of the contract.
`version`	`INT`	`NOT NULL`, `DEFAULT 1`	Version number of the contract.
`description`	`TEXT`	`NULL`	Detailed description of the contract.
`sender_partner_id`	`UUID`	`NOT NULL`, `FOREIGN KEY REFERENCES partners(partner_id)`	The partner ID of the sender in this contract.
`receiver_partner_id`	`UUID`	`NOT NULL`, `FOREIGN KEY REFERENCES partners(partner_id)`	The partner ID of the receiver in this contract.
`schema_definition`	`JSONB`	`NOT NULL`	JSONB field storing the data schema (e.g., JSON Schema).
`metadata`	`JSONB`	`NULL`	Additional contract metadata (purpose,

			security classification).
`validation_script_id`	`UUID`	`NULL`, `FOREIGN KEY REFERENCES validation_scripts(script_id)`	Optional: Reference to a validation script.
`retention_policy_days`	`INT`	`NULL`	Data retention period in days for this contract.
`pii_fields`	`TEXT[]`	`NULL`	Array of field names identified as PII.
`status`	`VARCHAR(50)`	`NOT NULL`, `DEFAULT 'Draft'`	Contract status: 'Draft', 'Proposed', 'Active', 'Terminated', 'Rejected'.
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the contract was created.
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the contract was last updated.

8. `data_transfers` Table

Records details of each data transfer event.

Column Name	Data Type	Constraints	Description
transfer_id	UUID	PRIMARY KEY, NOT NULL, DEFAULT gen_random_uuid()	Unique identifier for the data transfer.
contract_id	UUID	NOT NULL, FOREIGN KEY REFERENCES data_contracts(contract_id)	Foreign key to the associated data contract.
sender_node_id	VARCHAR(255)	NOT NULL	Node ID of the sending DX Node.
receiver_node_id	VARCHAR(255)	NOT NULL	Node ID of the receiving DX Node.
file_name	VARCHAR(255)	NOT NULL	Original name of the transferred file.
file_size_bytes	BIGINT	NOT NULL	Size of the transferred file in bytes.
file_checksum	VARCHAR(255)	NOT NULL	Checksum of the file content (e.g., SHA256).

sequence_number	BIGINT	NOT NULL	Monotonically increasing sequence number from sender.
message_timestamp	TIMESTAMP WITH TIME ZONE	NOT NULL	Timestamp assigned by the sender at message generation.
status	VARCHAR(50)	NOT NULL, DEFAULT 'Pending'	Transfer status: 'Pending', 'Processing', 'Sent', 'Received', 'Failed', 'Delivered'.
error_message	TEXT	NULL	Details if the transfer failed.
local_file_path	TEXT	NULL	Local path to the stored data file (for received/archived).
initiated_by_user_id	UUID	NULL, FOREIGN KEY REFERENCES users(user_id)	Optional: User who initiated the transfer.
start_time	TIMESTAMP WITH TIME ZONE	NOT NULL, DEFAULT NOW()	Timestamp when the transfer process started.

`end_time`	`TIMESTAMP WITH TIME ZONE`	`NULL`	Timestamp when the transfer process completed.

9. `audit_logs` Table

Records all significant events within the DX Node for auditing purposes.

Column Name	Data Type	Constraints	Description
`log_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the log entry.
`timestamp`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp of the event.
`event_type`	`VARCHAR(100)`	`NOT NULL`	Type of event (e.g., 'LOGIN_SUCCESS', 'TRANSFER_INITIATED', 'CONTRACT_UPDATED', 'REPLAY_DETECTED').
`severity`	`VARCHAR(50)`	`NOT NULL`, `DEFAULT 'INFO'`	Severity level: 'INFO', 'WARNING', 'ERROR', 'CRITICAL'.
`actor_type`	`VARCHAR(50)`	`NOT NULL`	Type of entity performing the action: 'User', 'System', 'PartnerNode'.

`actor_id`	`VARCHAR(255)`	`NOT NULL`	ID of the actor (user_id, node_id, or 'system').
`entity_type`	`VARCHAR(100)`	`NULL`	Type of entity affected (e.g., 'Partner', 'Contract', 'Transfer', 'User').
`entity_id`	`UUID`	`NULL`	ID of the affected entity.
`details`	`JSONB`	`NULL`	JSON object with additional event details.
`origin_ip`	`INET`	`NULL`	IP address from which the action originated.

10. `received_sequence_tracker` Table

Tracks the last successfully processed sequence number from each sender for replay protection.

Column Name	Data Type	Constraints	Description
`tracker_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the tracker entry.
`sender_node_id`	`VARCHAR(255)`	`NOT NULL`	Node ID of the sending DX Node.

`receiver_node_id`	`VARCHAR(255)`	`NOT NULL`	Node ID of this receiving DX Node (local node).
`contract_id`	`UUID`	`NOT NULL`, `FOREIGN KEY REFERENCES data_contracts(contract_id)`	The data contract associated with this sequence.
`last_received_sequence_number`	`BIGINT`	`NOT NULL`, `DEFAULT 0`	The highest sequence number successfully processed.
`last_received_timestamp`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT '1970-01-01 00:00:00Z'`	Timestamp of the last successfully processed message.
`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when this tracker entry was last updated.
		`UNIQUE (sender_node_id, receiver_node_id, contract_id)`	Ensures one tracker per sender-receiver-contract.

11. `pii_rules` Table

Stores rules for Personally Identifiable Information (PII) anonymization.

Column Name	Data Type	Constraints	Description
`rule_id`	`UUID`	`PRIMARY KEY`, `NOT NULL`, `DEFAULT gen_random_uuid()`	Unique identifier for the PII rule.
`field_name`	`VARCHAR(255)`	`NOT NULL`	The name of the data field to apply the rule to.
`anonymization_method`	`VARCHAR(50)`	`NOT NULL`	Method: 'HASH', 'MASK', 'TOKENIZE', 'REDACT'.
`config`	`JSONB`	`NULL`	JSON object for method-specific configurations (e.g., mask character, tokenization key ID).
`description`	`TEXT`	`NULL`	Description of the rule.
`created_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the rule was created.

`updated_at`	`TIMESTAMP WITH TIME ZONE`	`NOT NULL`, `DEFAULT NOW()`	Timestamp when the rule was last updated.

Indexes (Recommended):

- `CREATE INDEX idx_partners_node_id ON partners (node_id);`
- `CREATE INDEX idx_partners_fqdn ON partners (fqdn);`
- `CREATE INDEX idx_users_username ON users (username);`
- `CREATE INDEX idx_data_contracts_sender_partner_id ON data_contracts (sender_partner_id);`
- `CREATE INDEX idx_data_contracts_receiver_partner_id ON data_contracts (receiver_partner_id);`
- `CREATE INDEX idx_data_transfers_contract_id ON data_transfers (contract_id);`
- `CREATE INDEX idx_data_transfers_sender_node_id ON data_transfers (sender_node_id);`
- `CREATE INDEX idx_data_transfers_receiver_node_id ON data_transfers (receiver_node_id);`
- `CREATE INDEX idx_data_transfers_status ON data_transfers (status);`
- `CREATE INDEX idx_audit_logs_timestamp ON audit_logs (timestamp DESC);`
- `CREATE INDEX idx_audit_logs_event_type ON audit_logs (event_type);`
- `CREATE INDEX idx_audit_logs_actor_id ON audit_logs (actor_id);`
- `CREATE INDEX idx_received_sequence_tracker_sender_node_id ON received_sequence_tracker (sender_node_id);`

11. Security Architecture

Security is paramount in the TSI DX Node system.

- **Identity & Authentication:**
  - **TSI DX Nodes:** Mutual TLS (mTLS) using X.509 certificates for strong, bidirectional authentication between TSI DX Nodes. Certificates are issued against the Node's FQDN (CNAME).

    - **Users/Applications:** API Keys, JWTs, or OAuth2 for authentication with the TSI DX Node's API Gateway.
- **Authorization:**
    - **TSI DX Nodes:** Data contracts define which partners are authorized to send/receive specific data types.
    - **Users/Applications:** Role-Based Access Control (RBAC) governs user permissions to node functionalities (upload, download, configure, audit).
- **Confidentiality:**
    - **Data in Transit:** All P2P communication is encrypted using TLS 1.3.
    - **Data at Rest:** Data files on disk can be encrypted (OS-level encryption or application-level encryption for sensitive archives). Metadata in the database should be protected by database-level security.
    - **PII Anonymization:** Sensitive PII fields are transformed before transmission as per contract.
- **Integrity:**
    - **Message Hashing & Signing:** All messages are hashed and cryptographically signed by the sender's TSI DX Node private key. The receiver verifies this signature to ensure data has not been tampered with in transit.
    - **Schema Validation:** Ensures data conforms to agreed-upon structures.
    - **Audit Logs:** Tamper-proof audit logs provide an immutable record of all activities.
- **Availability:**
    - **Fault Tolerance:** Message queues, retries, and connection resilience mechanisms ensure transfers complete even with transient network issues.
    - **Replay Protection:** Prevents duplicate processing from unauthorized re-transmissions while allowing for legitimate re-sends.
- **Key Management:** Secure generation, storage, and rotation of private keys for TSI DX Nodes.

12. Scalability & Reliability

- **Scalability:**
    - **Initial Design (Single Reference Node):** The TSI DX Node is initially designed as a robust, single instance per organization, capable of handling typical data exchange volumes for small to medium-sized businesses.
    - **Enabling Extensions for Scalability:** The modular and API-first architecture, combined with the use of containerization (Docker) and asynchronous processing (Internal Message Queue), provides the foundation for partners to extend and add scalability aspects as needed.
        - Individual services (e.g., Data Transfer Engine, Data Processing Module) can be containerized and potentially scaled horizontally in a multi-instance deployment.

        - The choice of PostgreSQL for the metadata database supports high concurrency and can be scaled (e.g., read replicas, sharding) in more advanced setups.
        - The internal message queue (Kafka/RabbitMQ) is inherently scalable and can be deployed as a cluster to handle increased message throughput.
- **Reliability:**
    - **Message Queues:** Ensure no data transfer requests are lost due to transient failures.
    - **Retry Mechanisms:** Automatic retries for failed P2P connections or transfer attempts.
    - **Dead-Letter Queues:** Capture messages that consistently fail for manual inspection.
    - **Audit Logs:** Provide full traceability for debugging and incident response.
    - **Replay Protection:** Prevents integrity issues from duplicate messages.
    - **Configuration & State Replication (Advanced/Commercial):** For highly available, multi-node deployments, mechanisms for replicating node configuration and critical state across instances would be implemented by partners or commercial offerings.

13. Deployment Model

The TSI DX Node is designed for flexible deployment within an organization's existing IT infrastructure.

- **Initial Deployment (Single Reference Node):** The recommended initial deployment is a single instance of the TSI DX Node application, running on:
    - **On-Premise:** Directly on physical servers or virtual machines (VMs) running Linux or Windows Server.
    - **Containerized:** Using Docker containers, managed by Docker Compose for simplified single-node setup and management.
- **Scalable Deployments (Partner Extensions):** Partners can extend this foundational setup for higher availability and scalability by deploying multiple instances of specific modules (e.g., Data Transfer Engine workers) using:
    - **Container Orchestration:** Kubernetes for clustered, highly available environments.
    - **Cloud:** Easily deployable on any cloud provider's VM or container services (e.g., AWS EC2/ECS/EKS, Azure VMs/AKS, GCP Compute Engine/GKE) in a single-instance or scaled configuration.

Each organization manages its own TSI DX Node instance(s), including its hardware/VM resources, network configuration (DNS CNAME, firewall rules), and software updates.

14. Project Plan

Estimated Duration: 24 Weeks (Approx. 6 months)

Phase 1: Foundation & Core Infrastructure (Weeks 1-3)

- **Objective:** Rapidly set up the core development environment, establish foundational services, and implement basic node identity.
- **Key Activities:**
  - **Project Kick-off & Team Alignment:** (Week 1) Finalize roles, communication, agile ceremonies.
  - **Detailed Design Review & Refinement:** (Weeks 1-2) Rapid review of SAD, API, DB schema; identify any immediate blockers.
  - **Technology Stack Setup:** (Weeks 1-2) Choose primary language/frameworks. Set up PostgreSQL, Message Queue (Kafka/RabbitMQ) instances. Establish Docker/Containerization.
  - **Database Initialization:** (Week 2) Implement `init.sql` and initial migration scripts.
  - **Basic Node Service Skeleton:** (Weeks 2-3) Create main application structure, configuration loading, and logging.
  - **Basic API Gateway:** (Week 3) Implement core HTTP server, request routing, and initial local admin login.
- **Deliverables:** Configured dev environment, initialized DB, runnable DX Node skeleton, basic API Gateway.

Phase 2: Identity, Partner Management & Security Core (Weeks 4-7)

- **Objective:** Enable the node to establish its identity, securely manage partners, and secure communications.
- **Key Activities (Parallel Execution):**
  - **Backend (Weeks 4-7):**
    - **PKI Implementation:** Develop modules for CSR generation, certificate/key import, secure storage. Implement mTLS handshake logic.
    - **Partner Entity Management:** Implement CRUD for `partners` via Admin API. Develop basic partner discovery. Implement `ReceivedSequenceTracker` logic.
    - **Initial Audit Logging:** Integrate basic logging for user actions and critical system events.
  - **Frontend (Weeks 5-7 - Start in parallel with backend):**
    - Begin Admin UI for Partners: Develop UI for adding, viewing, editing partners.
- **Deliverables:** DX Node capable of mTLS. Functional Partner Management (API & initial UI). Core audit trail.

Phase 3: Data Contracts & Core Transfer Logic (Weeks 8-12)

- **Objective:** Implement the core data contract definition and the fundamental peer-to-peer data transfer mechanism with replay protection.
- **Key Activities (Parallel Execution):**
    - **Backend (Weeks 8-12):**
        - **Data Contract Management:** Implement CRUD for `data_contracts` (schema, metadata). Develop contract lifecycle (propose, accept, reject, terminate). Integrate `validation_scripts` table.
        - **Data Transfer Engine (Core):** Implement single-file P2P send/receive. Integrate with `data_contracts` for pre-transfer validation. Implement `sequence_number`/`message_timestamp` generation and **replay protection checks**.
    - **Frontend (Weeks 9-12 - Continue parallel with backend):**
        - Develop Admin UI for Data Contracts: Create, manage, view contracts.
        - Develop Admin UI for Basic Transfers: Initiate single transfers, view status.
- **Deliverables:** Functional Data Contract management. Secure, replay-protected single-file data transfer. Core Admin UI for contracts and transfers.

Phase 4: Advanced Governance & Transfer Features (Weeks 13-18)

- **Objective:** Enhance the node with advanced data governance, bulk transfer, and robust access control.
- **Key Activities (High Parallelization):**
    - **Backend (Weeks 13-18):**
        - **PII Anonymization Engine:** Implement configurable PII anonymization methods.
        - **Validation Scripting Engine:** Integrate runtime for Python/JavaScript, sandboxing.
        - **Bulk Data Upload:** Implement backend logic for `bulk_uploads`.
        - **Legitimate Re-send:** Implement logic for new sequence numbers on re-sends.
        - **Data Archiving & Purging:** Develop logic for retention policies and secure deletion.
        - **Role-Based Access Control (RBAC):** Implement `users`, `roles`, `role_permissions` logic.
        - **Client API Development:** Implement `POST /client/transfers/initiate`, `GET /client/transfers/{id}/status`, `GET /client/transfers/received`, `GET /client/transfers/received/{id}/download`.
    - **Frontend (Weeks 13-18 - Continue parallel with backend):**
        - Develop Admin UI for Governance (PII rules, validation scripts, archiving/purging).

        - Develop Admin UI for Users & Roles (managing users, defining roles/permissions).
        - Integrate bulk upload UI.
- **Deliverables:** Full data governance features. Bulk transfer & re-send. Comprehensive RBAC. Functional Client API.

Phase 5: UI Completion, Integration & Hardening (Weeks 19-24)

- **Objective:** Finalize the Admin UI, conduct rigorous testing, and prepare for open-source release.
- **Key Activities (Intensive Parallelization):**
    - **Admin UI Completion:** (Weeks 19-21)
        - Develop Dashboard.
        - Enhance Data Transfers UI (detailed views, filtering).
        - Implement comprehensive Audit Logs UI (search, filter, export).
        - Refine all UI for usability and responsiveness.
    - **Integration Testing:** (Weeks 19-22) End-to-end testing of all modules and API interactions.
    - **Security Testing:** (Weeks 20-22) Penetration testing, vulnerability scanning, code audits.
    - **Performance Testing:** (Weeks 21-22) Load testing for reference node.
    - **Bug Fixing & Refinement:** (Weeks 19-23) Continuous bug fixing.
    - **Deployment Packaging:** (Weeks 22-23) Create production-ready Docker images, example Docker Compose.
    - **Comprehensive Documentation:** (Weeks 19-24) Write user, admin, developer, and contribution guides. **This must be done concurrently with development.**
    - **Open-Source Release Preparation:** (Week 24) Finalize licensing, set up GitHub repository, define community guidelines.
- **Deliverables:** Feature-complete, stable, and secure TSI DX Node. Comprehensive test reports. Production-ready software package. Complete documentation. Public GitHub repository.