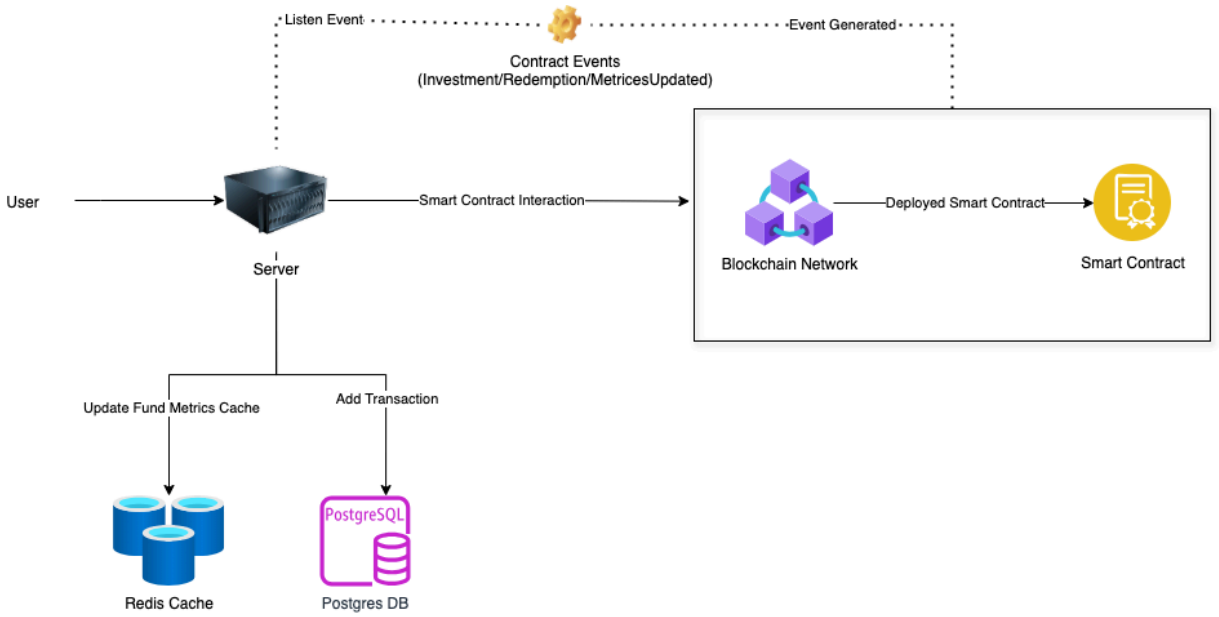
System Design Document

1. Overview

This document outlines the system architecture for an Investment Fund Management System that utilizes blockchain technology to manage investments and redemptions. The system interfaces with a smart contract, handles transactions, tracks fund metrics for caching, monitors Investment, Redemption and MetricsUpdated events, stores data in a database, and ensures data consistency between the blockchain and the backend database.

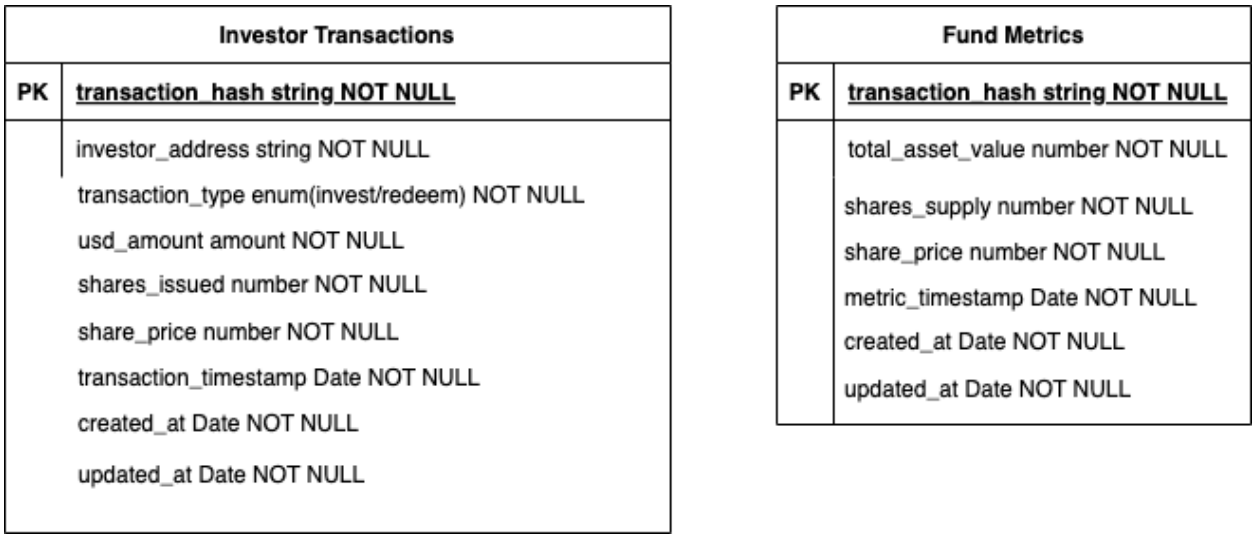
2. Architecture Diagram



User: Client application.
Server: Backend service using Node.js and TypeScript.
Blockchain Network: Contains the deployed smart contract (EVM).
Smart Contract: Manages core logic for Investment fund operations.
PostgreSQL Database: Stores transaction records.
Redis Cache: Caches fund metrics for high-performance data access.

3. Data Models

PostgresSQL Data Model:



Investor Transactions Table:

- **transaction_id (Primary Key, UUID):** Unique identifier for each transaction.
- **investor_address (String):** Blockchain address of the investor.
- **transaction_type (Enum: 'invest', 'redeem'):** Type of the transaction.
- **usd_amount (Decimal):** USD amount involved in the transaction.
- **shares_issued (Int):** Number of shares involved.
- **share_price (Decimal):** Price per share at the time of the transaction.
- **timestamp (Timestamp):** Date and time the transaction was processed.

Fund Metrics:

- **transaction_id (Primary Key, UUID):** Unique identifier for each transaction.
- **total_asset_value (decimal):** Total value of the asset in USD.
- **shares_supply (int):** Total number of shares issued.
- **share_price (decimal):** Share price in USD.

Redis Data Model:

Fund Metrics	
	totalAssetValue number NOT NULL
	sharesSupply number NOT NULL
	lastUpdateTime Date NOT NULL

Fund Metrics: Stores a JSON object.

- **total_asset_value (decimal):** Total value of the asset in USD.
- **shares_supply (int):** Total number of shares issued.
- **last_update_time (timestamp):** Timestamp of the last update to metrics.

4. Component Interactions

4.1 Investment Flow

- **User** initiates an investment by sending a request to the **Server**.
- **Server** validates the request and sends a transaction to the **Blockchain Network** to invoke the `invest` method on the **Smart Contract**.
- **Smart Contract** processes the transaction, updates the state, and emits an **Investment event**.
- **Server** listens for the **Investment event**, then adds the transaction details to the **PostgreSQL Database**.
- **Server** sends a response back to the **User** with the transaction results, confirming the successful investment.

4.2 Redemption Flow

- **User** requests a redemption, which is received by the **Server**.
- **Server** validates the request and sends a transaction to the **Blockchain Network** to invoke the `redeem` method on the **Smart Contract**.
- Upon successful transaction processing, the **Smart Contract** emits a **Redemption event**.
- **Server** captures the event, adds the transaction details to the **PostgreSQL Database**.

- **User** receives the transaction outcome from the **Server**, confirming the successful redemption.

4.3 Event Monitoring and Data Handling

- **Server** continuously monitors for events emitted by the **Smart Contract**, such as `Investment`, `Redemption`, and `MetricsUpdated`.
- Upon detecting any of these events, the **Server** will update the transaction history in the **PostgreSQL Database**.

5. Summary

The system design utilizes Node.js with TypeScript for a robust and scalable backend environment, PostgreSQL for persistent data storage, and Redis for caching to enhance performance. This architecture ensures efficient management of blockchain interactions, reliable data storage, and quick access to frequently needed data. The design is focused on providing a secure, reliable, and user-friendly platform for investment fund management.