**Architecture Design Document (ADD) for USDC Transfer Aggregation Service**

**1. Introduction**

**1.1 Purpose**

This document outlines the architectural design for a **backend service** that interacts with the **Avalanche Blockchain** to fetch, aggregate, and analyze **USDC (a stablecoin)** transfer data in real time. The service processes the data from Blockchain transactions, stores it in a database, and provides various insights via API endpoints.

**1.2 Scope**

The service provides functionality to:

* Fetch USDC transactions from the Avalanche Blockchain.
* Store transaction data in a **PostgreSQL** database.
* Aggregate data and provide insights such as:
  + Total USDC transferred in a specified time range.
  + USDC transfers in a specified time range.
  + Top accounts by transaction volume.
* Provide **RESTful API** endpoints to query aggregated data.
* Implement robust error handling and validation to ensure service reliability.

**1.3 Intended Audience**

This document is aimed at:

* **Client Stakeholders**
* **System Architects**
* **Project Managers**

**1.4 Definitions and Acronyms**

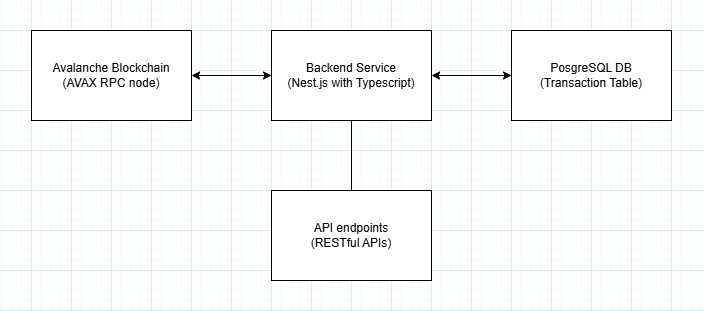
* **USDC**: USD Coin, an ERC-20 stablecoin.
* **AVAX**: A RPC node for Avalanche Blockchain.
* **RPC**: Remote Procedure Call, used to interact with the Avalanche Blockchain.
* **NestJS**: A Node.js framework for building efficient and scalable server-side applications using TypeScript.
* **PostgreSQL**: A powerful, open-source relational database system.
* **TypeORM**: A TypeScript ORM for working with databases in NestJS.

**2. System Overview**

The backend service is designed to:

* Connect to the **Avalanche Blockchain** via an AVAX RPC node.
* Fetch real-time USDC transactions from the Blockchain and aggregate this data.
* Store the data in a **PostgreSQL database**.
* Expose RESTful APIs for querying aggregated USDC transfer data.

**High-Level Architecture Diagram**

  
**Key Components:**

* **Backend Service (NestJS)**: Handles business logic, interacts with the Avalanche Blockchain using AVAX RPC node, aggregates data, and serves the API.
* **PostgreSQL Database**: Stores transaction data and allows querying for aggregation and insights.
* **API Endpoints**: Expose various API routes for querying aggregated data (e.g., total transferred, top accounts).

**3. Functional Requirements**

**3.1 Data Fetching and Storage:**

* **Initial USDC Transaction Fetch**:
  + The backend service fetches all USDC transactions from the **Avalanche Blockchain** starting from the beginning of the chain.
  + This historical data is stored in the **PostgreSQL database**.
* **Real-time Data Fetch**:
  + The service continuously fetches new **USDC transfer events** in real-time usi
  + ng the AVAX RPC node.
  + These events are stored in the PostgreSQL database as they are received.

**3.2 API Endpoints:**

* **getTotalTransferred()**:
  + API that returns the total amount of USDC transferred within a specified period.
* **getUSDCTransfers()**:
  + Query endpoint to fetch all USDC transfers within a given block range (from fromBlock to toBlock).
* **getTopAccounts()**:
  + Query endpoint that returns top accounts by transaction volume within a given block range.
* **getMonthlyTotal()**:
  + Query endpoint that returns monthly total amount of USDC transferred within a year backward from given block time. (example of timestamp: 9999999999)

**3.3 Error Handling and Data Validation:**

* **Error Handling**:
  + Handle potential issues such as failed Blockchain interaction, missing data, or malformed requests.
  + Return appropriate HTTP error codes (e.g., 400 for bad requests, 500 for server errors).
* **Data Validation**:
  + Ensure that transaction data fetched from the Blockchain is valid (e.g., non-negative transfer amounts, valid addresses).

**4. Architectural Design**

**4.1 Blockchain Integration:**

* **AVAX RPC Node**:
  + The backend communicates with the Avalanche Blockchain via AVAX RPC to fetch **USDC transactions** (ERC-20 transfers on the C-Chain).
  + The service uses **ethers.js** to query the Blockchain for transfer events.

**4.2 Backend (NestJS):**

* **NestJS Framework**:
  + Handles incoming requests, processes Blockchain data, aggregates information, and serves API responses.
  + Uses **TypeORM** to interact with the PostgreSQL database and manage migrations.
* **Data Aggregation**:
  + The service processes USDC transaction data to provide aggregated insights, such as total USDC transferred and top transaction accounts.
  + These aggregations can be done in real-time and on-demand based on API queries.

**4.3 Database (PostgreSQL):**

* **PostgreSQL Database**:
  + Stores the following transaction data:
    - id: Primary key.
    - transaction\_hash: Unique hash for each transaction.
    - sender\_address: Address of the sender.
    - receiver\_address: Address of the receiver.
    - amount: Amount of USDC transferred.
    - timestamp: Transaction timestamp.
* **Database Migrations**:
  + **TypeORM** is used to define and manage database migrations.

**4.4 API Design:**

* **getTotalTransferred()**:
  + Parameters: startDate, endDate (Time range for the data aggregation).
  + Returns: The total USDC transferred between the specified dates.
* **getUSDCTransfers()**:
  + Parameters: fromBlock, toBlock (Block range for querying transfers).
  + Returns: A list of USDC transactions in the specified block range.
* **getTopAccounts()**:
  + Parameters: startDate, endDate, limit (Limit the number of top accounts returned).
  + Returns: A list of top accounts based on transaction volume.
* **getMonthlyTotal()**:
  + Parameters: endDate (End time for the monthly statistics).
  + Returns: A list of USDC amount for each months.

**5. Data Flow**

1. **Initial Data Fetch**:
   * The backend service connects to the AVAX RPC node and queries the Blockchain for **all historical USDC transfers**.
   * This data is then stored in the PostgreSQL database.
2. **Real-time Data Fetch**:
   * The service listens for new **USDC transfer events** from the Avalanche Blockchain.
   * As new transactions are received, they are stored in the database.
3. **API Data Aggregation**:
   * When a request is made to the API endpoints, the backend fetches and aggregates data from the database.
   * Data is returned in the appropriate format, based on the endpoint and query parameters.

**10. Conclusion**

This document provides a detailed overview of the **architecture design** for the USDC transfer aggregation service. The service is designed to interact with the **Avalanche Blockchain**, store and aggregate transaction data, and expose insightful API endpoints. By using **NestJS**, **PostgreSQL**, and Blockchain integration, the service ensures reliable, real-time data handling with robust error handling and performance considerations.