Documentation : - Write comprehensive documentation explaining the flow distribution algorithm, its design principles, and how to interact with the API endpoints.

**Overview**

This project implements a flow distribution algorithm in Node.js to connect users with astrologers in a fair and balanced manner. The algorithm ensures each astrologer gets an equal proportion of chances to connect with users while providing the flexibility to adjust the flow for top astrologers.

**Requirements**

1. **Fair Allocation**: Allocate users to astrologers in a balanced manner.
2. **Adjustable Flow**: Allow adjustment of flow for top astrologers.
3. **Scalability**: Efficiently handle 2000-3000 users daily with a pool of 500 astrologers.
4. **Documentation**: Provide clear documentation for the algorithm and its usage.
5. **Testing**: Validate the algorithm with test cases.

**Algorithm Design Principles**

1. **Fair Distribution**:
   * Each astrologer should receive an equal number of user connections relative to their flow rate.
   * The flow rate represents the relative share of connections an astrologer should receive.
2. **Load Balancing**:
   * The algorithm balances the load by tracking the number of current connections each astrologer has.
   * Users are allocated to astrologers with the lowest load adjusted by their flow rate.
3. **Flow Rate Adjustment**:
   * The algorithm allows dynamic adjustment of an astrologer's flow rate.
   * This provides flexibility to favor certain astrologers by increasing their chances of getting more connections.

**Data Structures**

**Astrologer**

* **id**: Unique identifier.
* **name**: Name of the astrologer.
* **flowRate**: Number representing the flow preference (default is 1).
* **currentLoad**: Number of current user connections.

**User**

* **id**: Unique identifier.
* **name**: Name of the user.

**Algorithm Implementation**

**FlowDistribution Class**

* **Constructor**: Accepts an array of astrologers and initializes the distribution system.
* **getAstrologerWithMinLoad**: Finds the astrologer with the minimum load adjusted by flow rate.
* **distributeUser**: Allocates a user to the astrologer with the minimum load.
* **setTopAstrologerFlowRate**: Adjusts the flow rate for a specified astrologer.

### API Endpoints

#### Add an Astrologer

**Endpoint**: POST /api/astrologers

**Request Body**: { "id": 1, "name": "Astrologer A", "flowRate": 1 }

**Response:**

{ "id": 1, "name": "Astrologer A", "flowRate": 1, "currentLoad": 0 }

#### Distribute a User to an Astrologer

**Endpoint**: POST /api/users

Request Body: { "id": 1, "name": "User A" }

Response : { "user": { "id": 1, "name": "User A" }, "astrologer": { "id": 1, "name": "Astrologer A", "flowRate": 1, "currentLoad": 1 } }

#### Set Flow Rate for a Top Astrologer

**Endpoint**: PUT /api/astrologers/:id/flowRate

Request Body: { "flowRate": 2 }

Response: { "id": 1, "flowRate": 2 }

**Detailed Steps**

1. **Initialize Data Structures**:
   * Define classes for Astrologer and User.
   * Define the FlowDistribution class with methods for load balancing and flow rate adjustment.
2. **Set Up Express Server**:
   * Create the Express application in app.js.
   * Define routes in api.js for adding astrologers, distributing users, and setting flow rates.
   * Set up the server in server.js.
3. **Implement Algorithm**:
   * Implement the load balancing and flow rate adjustment logic in flowDistribution.js.
4. **Testing**:
   * Write test cases in flowDistribution.test.js to validate the algorithm under different scenarios.

**Testing**

To run the tests, use the following command: npm test

### Running the Server

To start the server, use the following command: npm start

The server will run on port 3000 by default.