FinTarget Task Documentation

Overview

This application implements a RESTful API using Express.js to manage user tasks with a rate-limiting mechanism. The API allows users to submit tasks, ensuring that each user can only create a limited number of tasks within a specified timeframe. The application is designed for scalability by utilizing Node.js clustering to take advantage of multi-core systems.

Key Features

1. Rate Limiting: The API restricts each user to a maximum of 20 tasks per minute and 1 task per second.
2. Task Logging: Completed tasks are logged to a file for tracking and auditing purposes.
3. Task Queueing: Requests that exceed the rate limit are queued and processed once the limit allows.
4. Clustered Architecture: The application uses Node.js clustering to handle multiple requests concurrently, improving performance and responsiveness.

Setup and Dependencies

Required Libraries

- express: A minimal web framework for Node.js.
- fs: Node.js file system module for logging task completions.
- rate-limiter-flexible: A flexible rate limiter for managing request limits.
- cluster: Node.js module for creating child processes (workers) for concurrency.
- os: Node.js module for operating system-related utility functions.
- cors: Middleware to enable Cross-Origin Resource Sharing.

Installation

To install the necessary dependencies, run:

npm install express rate-limiter-flexible cors

API Endpoints

POST /task

- Description: Submits a task for processing.
- Request Body:
    - user_id: (string) The ID of the user submitting the task. This field is required.
- Responses:
    - 202 Accepted: Task has been queued and processed immediately.
    - 400 Bad Request: Missing user_id.
    - 429 Too Many Requests: Rate limit exceeded; the request will be processed later.

Rate Limiting

The rate limiter is configured with the following parameters:

- Points: 20 (the maximum number of tasks a user can submit within the duration).
- Duration: 60 seconds (the period in which the points are calculated).

If a user exceeds this limit, their request is queued, and the application logs the event to a file named rate_limit_log.txt.

Task Processing Logic

Task Functionality

1. Logging: Upon successful task completion, the application logs the user_id and timestamp to task_log.txt.
2. Database Interaction: Each completed task is stored in a database using a Task model, which is assumed to be defined elsewhere in the application.

Queuing System

- When a request is made, the system checks if the user is already in the pendingRequests queue. If not, it initializes an empty queue for them.

- If the rate limit is not exceeded, the task is processed immediately.
- If the limit is exceeded, the request timestamp is pushed onto the user's queue, and the request is responded to with a 429 status code.
- A separate function (processPendingRequests) processes the queued requests at a fixed interval (1 second), respecting the rate limit.

Clustering

The application utilizes the Node.js cluster module to fork multiple worker processes, enabling concurrent handling of incoming requests. Each worker listens on the same port, allowing the application to leverage multiple CPU cores.

Master Process

- The master process forks a number of workers equal to the number of CPU cores available on the machine.
- It also listens for worker exit events to log any worker failures.

Example Usage

To submit a task, send a POST request to /task with a JSON body containing the user_id:

{

"user_id": 123

}

Conclusion

This task processing API is designed to efficiently manage user tasks with robust rate-limiting and logging features, making it suitable for applications requiring high concurrency and task tracking. The implementation demonstrates best practices in Node.js development, including middleware usage, asynchronous programming, and system resource management through clustering.