Node.js Task Queuing with Rate Limiting

Project Overview

This project is a Node.js-based API that handles task requests with rate limiting and task queuing. It ensures that tasks are processed at a controlled rate per user ID, adhering to specific rate limits. The project also includes a front-end component built using React.js for monitoring and interacting with the API.

Technologies Used

Backend (Node.js)

- **Node.js**: The core runtime used to build the API.
- **Express.js**: A web framework for Node.js used to create the API endpoints.
- **Redis**: Used for implementing the task queueing system, providing in-memory data structure storage and messaging.
- **Cluster Module**: Utilized to set up Node.js clusters to handle multiple processes and distribute the load across CPU cores.
- **Rate Limiter**: Custom middleware built to enforce rate limits on API requests.
- **File System (fs) Module**: Used to log task completion data into a `task_logs.txt` file.

Frontend (React.js)

- **React.js**: Used to create a simple user interface for interacting with the API.
- **Axios**: A promise-based HTTP client used to send requests to the backend API.

Project Structure

```
node-task-queuing/
│
├── src/
│   ├── config/
│   │   └── redisConfig.js
│   ├── controllers/
│   │   └── taskController.js
│   ├── logs/
│   │   └── task_logs.txt
│   ├── middlewares/
│   │   └── rateLimiter.js
│   ├── utils/
│   │   └── taskQueue.js
│   ├── routes/
│   │   └── taskRoutes.js
│   ├── app.js
│   └── cluster.js
├── package.json
├── README.md
└── node-task-queuing-ui/
    ├── public/
    ├── src/
    │   ├── App.js
    │   └── index.js
    ├── package.json
    └── README.md
```

Key Files and Their Roles

- **`app.js`**: The main entry point of the application. It initializes the Express server and sets up the routes.
- **`cluster.js`**: Manages the clustering of the Node.js application, creating worker processes.
- **`redisConfig.js`**: Configuration for connecting to the Redis server.
- **`taskController.js`**: Handles the task processing logic, including logging task completions.
- **`rateLimiter.js`**: Middleware that enforces the rate limits for the API.
- **`taskQueue.js`**: Utility for managing the task queue using Redis.
- **`taskRoutes.js`**: Defines the API route for task handling.
- **`task_logs.txt`**: The log file where task completion data is stored.

How the Project Works

1. Task Submission

Users can submit tasks via a POST request to the `/api/v1/task` endpoint, including their user ID in the request body.

2. Rate Limiting

The rate limiting middleware ensures that each user ID can only submit one task per second and no more than 20 tasks per minute. Any requests exceeding this limit are queued.

3. Task Queueing

Tasks that exceed the rate limit are placed in a queue. The queue is managed using Redis, which ensures tasks are processed in the order they were received.

4. Task Processing

The `taskController.js` handles processing tasks from the queue. Once a task is processed, it is logged into the `task_logs.txt` file with the user ID and timestamp.

5. Clustering

The Node.js application uses clustering to handle multiple processes, improving the application's ability to handle concurrent tasks.

How to Run the Project

Prerequisites

- **Node.js** and **npm** installed on your machine.
- **Redis** server running locally or accessible via the configured connection.

Steps to Run

1.

Install Dependencies

```
npm install
cd node-task-queuing-ui
npm install
```

2.

Start the Redis Server Ensure your Redis server is running. You can start it using:

```
redis-server
```

3.

Run the Backend Server Go back to the main directory:

```
cd ..
npm run start
```

4.

Run the Frontend

```
cd node-task-queuing-ui
npm start
```

5.

Test the API Use Postman or a similar tool to send a POST request to `http://localhost:3000/api/v1/task` with the following JSON body:

```
{
   "user_id": "123"}
```

Example:

```
curl -X POST http://localhost:3000/api/v1/task -H "Content-Type:
application/json" -d '{"user_id": "123"}'
```

6. **Check Task Logs** Task completions are logged in the `src/logs/task_logs.txt` file.