Backend System Design Using Queue

Objective

Design and implement a backend system that efficiently manages requests from multiple users using a queue structure. Each client connected will have its own queue where all requests are processed sequentially. The system will be robust and scalable, ensuring that queues are emptied once all requests are processed and all users disconnect.

Requirements

1. **User Authentication**: Securely authenticate users before they can enqueue requests.
2. **Request Queueing**: Implement a queue for each client to handle requests in a First-In-First-Out (FIFO) manner.
3. **Request Processing**: Develop a process to handle and execute requests sequentially.
4. **Concurrency Management**: Handle multiple clients and their queues concurrently.
5. **Scalability**: Ensure the system can scale to handle an increasing number of users and requests without performance degradation.
6. **Robustness**: Implement error handling and recovery mechanisms to manage failures without data loss.
7. **Logging and Monitoring**: Set up logging for tracking request handling and system monitoring for performance metrics.

Tools and Technologies

- **Programming Language**: Node.js
- **Messaging/Queueing System**: RabbitMQ / Redis / Kafka
- **Database**: PostgreSQL / MongoDB
- **Monitoring Tools**: Prometheus, Grafana

System Design

1. Client-Server Model

Users interact with the system through a client interface that sends requests to the server.

2. Queue Management

Each client connection has a dedicated queue. A queue manager handles the creation, management, and deletion of queues.

3. Worker Processes

Dedicated worker processes pull requests from queues and execute them sequentially.

Assignment Tasks

1. System Architecture

Draw a system architecture diagram showing:

- Client-server interaction.
- Queue system.
- Database integration.
- Worker processes.

Architecture Diagram Description

The system consists of:

- **Client Interface**: Sends requests to the backend.
- **Server**: Manages authentication, queue allocation, and request delegation.
- **Queue Manager**: Dynamically creates and manages queues for each client.
- **Worker Processes**: Processes requests sequentially from the queues.
- **Database**: Stores user data, request logs, and system metrics.
- **Monitoring Tools**: Tracks performance and logs errors.

2. Implementation

a. User Authentication

- Securely authenticate users via JWT or session-based authentication.

b. Queue Setup

- Implement a queue for each user using RabbitMQ, Redis, or Kafka.

c. Worker Processes

- Develop worker processes to pull requests from the queue and process them sequentially.

d. Logging

- Use Winston or a similar library for logging requests and errors.

e. Monitoring

- Integrate Prometheus for monitoring and Grafana for visualizing metrics.

3. Testing

- Write unit tests using Jest or Mocha to verify:
    - User authentication.
    - Queue creation and processing.
    - Request execution and logging.
    - Error handling and recovery.

4. Deployment

Prepare Docker containers for:

- Backend server.
- Queueing system.
- Worker processes.
- Database.
- Monitoring tools.

Dockerfiles

- Create Dockerfiles for each component, ensuring modularity and ease of scaling.

Flow Diagrams

1. Overall System Flow

Diagram showing:

- User sends a request to the server.
- Server authenticates the user.
- Request is enqueued into the user's dedicated queue.
- Worker processes handle requests sequentially.
- Results are logged, and metrics are updated.

2. Detailed Process Flow

Diagram illustrating:

- Steps from request reception to completion.
- Error handling and retry mechanisms.
- Monitoring and logging workflows.

Running the System

With Docker

1. **Prerequisites**:
   - Docker Desktop installed and running.
   - All necessary Dockerfiles and docker-compose.yml files created and placed in their respective directories.
2. **Steps**:
   - **Build Docker Images**:

```
docker build -t nodejs-queue-app .
```

Ensure you are in the project directory where the Dockerfile is located (e.g., the root of your project).

   - **Start the System**: If using docker-compose:

```
docker-compose up
```

Otherwise, run individual containers:

```
docker run -d -p 3000:3000 --name queue-backend nodejs-queue-app
```

   - **Verify**: Visit http://localhost:3000 to check the application or test the endpoints using tools like Postman.
   - **Stop the Containers**:

```
docker-compose down
```

Or if not using docker-compose:

```
docker stop queue-backend
```

```
docker rm queue-backend
```

3. **Environment Variables**:
   Set environment variables using a .env file in the project root and specify them in the Dockerfile or docker-compose.yml.

Without Docker

1. **Prerequisites**:
   - Node.js (v16+ recommended) and npm installed.
   - RabbitMQ/Redis/Kafka installed locally or hosted.
   - Database (MongoDB/PostgreSQL) installed and running locally or hosted.
2. **Steps**:
   - **Install Dependencies**:

```
npm install
```

   - **Set Up Environment Variables**:
     Create a .env file in the root directory with the following keys (update values as needed):

```
PORT=3000

DATABASE_URL=mongodb://localhost:27017/queue_system

QUEUE_SYSTEM=redis://localhost:6379

JWT_SECRET=your_jwt_secret_key
```

   - **Start RabbitMQ/Redis/Kafka**: Start the queueing system service you are using:
     - RabbitMQ: Use the RabbitMQ management console or CLI.
     - Redis: Start Redis with the command:

```
redis-server
```

   - **Start the Database**: Start your MongoDB/PostgreSQL instance:

```
mongod
```

   - **Run the Application**:

```
npm start
```

Or, for development mode:

```
npm run dev
```

   - **Verify**: Open http://localhost:3000 to test the application or use tools like Postman.

3. **Testing**: Run tests to ensure all components function as expected:

```
npm test
```

Additional Notes

- **RabbitMQ/Redis/Kafka**: If hosted remotely, update the .env file with the remote URL.
- **Database**: Use a cloud-hosted MongoDB/SQL database like Atlas or Heroku PostgreSQL if not running locally.
- **Monitoring**: Ensure Prometheus and Grafana are configured if you plan to monitor the system performance.