**Backend System Documentation**

**1. System Architecture**

**Overview**

The system is designed to manage requests from multiple users efficiently using a queue structure. The architecture includes components for user authentication, request handling, and monitoring.

**Detailed Diagrams**

**Overall System Flow Diagram**

+-------------+ +---------------+ +----------------+ +-------------+

| | | | | | | |

| Users | <---> | Client | <---> | Server (Node.js)| <---> | RabbitMQ |

| | | Interface | | (Express) | | Queue |

+---------------+ +----------------+ +--

| ^

|

v

+-----------------+

| |

| MongoDB |

| |

+-----------------+

|

v

+-----------------

| Worker

| ProcessRequest

+-----------------

**Detailed Process Flow Diagram**

plaintext

Copy code

+-------------+ +----------------+ +------------------+ +----------------+ +---------------------+

| | | | | | | | | |

| User | | Client | | Server (Node.js)| | RabbitMQ | | Worker Process |

| Request | ----> | Interface | -----> | Authentication | -----> | Queue | ----> | Process Requests |

| | | | | & Authorization| | | | & Update MongoDB |

+-------------+ +----------------+ +------------------+ +----------------+ +---------------------+

|

v

+----------------------------+

| |

| Request Queuing and |

| Enqueue in User-specific |

| Queue |

+----------------------------+

|

v

+---------------------------+

| |

| Request Processing |

| & Task Execution |

| |

+---------------------------+

|

v

+---------------------------+

| |

| Logging and Metrics |

| (Prometheus) |

| |

+---------------------------+

**Explanation**

* **Users**: Interact with the system through a client interface.
* **Client Interface**: Sends requests to the server.
* **Server (Node.js)**: Handles authentication, authorization, and manages the RabbitMQ queue.
* **RabbitMQ**: Queues user requests and directs them to worker processes.
* **Worker Process**: Executes requests sequentially and updates MongoDB.
* **MongoDB**: Stores user data and request details.
* **Prometheus**: Monitors system metrics, integrated with Grafana for visualization.

**2. Code Explanation**

**Components**

1. **User Authentication** (routes/auth.js, models/User.js):
   * **routes/auth.js**: Handles user registration and login. Generates JWT tokens for authenticated users.
   * **models/User.js**: Defines the user schema and interactions with MongoDB.
2. **Queue Management** (queue/queueManager.js):
   * Manages the creation of user-specific queues in RabbitMQ.
   * Enqueues and dequeues requests, ensuring FIFO processing.
3. **Worker Processes** (worker/worker.js):
   * Continuously listens to RabbitMQ queues.
   * Processes tasks and updates MongoDB with results.
4. **Monitoring** (monitoring/monitoring.js, routes/metrics.js):
   * **monitoring/monitoring.js**: Collects metrics for Prometheus.
   * **routes/metrics.js**: Provides an endpoint for Prometheus to scrape metrics.

**Setup and Interaction**

1. **Server**:
   * Starts on port 3000.
   * Listens for requests and performs authentication/authorization.
   * Interacts with RabbitMQ to manage queues.
2. **Queue Management**:
   * Uses RabbitMQ for queueing user requests.
   * Each user has a dedicated queue named queue\_userID.
3. **Worker Process**:
   * Pulls requests from RabbitMQ and processes them.
   * Updates MongoDB and logs results.
4. **Monitoring**:
   * Prometheus collects metrics from the metrics endpoint.
   * Grafana visualizes metrics for monitoring.

**3. Deployment Instructions**

**Using Docker**

1. **Dockerfile for Node.js Backend**

**Dockerfile**:

dockerfile

# Dockerfile for Node.js Backend

FROM node:14

# Create app directory

WORKDIR /usr/src/app

# Install dependencies

COPY package\*.json ./

RUN npm install

# Copy source files

COPY . .

# Expose port

EXPOSE 3000

# Start the application

CMD ["node", "index.js"]

1. **Docker-Compose File**

**docker-compose.yml**:

yaml

Copy code

version: '3'

services:

backend:

build: .

container\_name: node\_backend

ports:

- "3000:3000"

depends\_on:

- mongodb

- rabbitmq

environment:

- MONGO\_URI=mongodb://mongodb:27017/backend\_assignment

- RABBITMQ\_URI=amqp://rabbitmq

- JWT\_SECRET=your\_jwt\_secret

networks:

- backend\_network

mongodb:

image: mongo

container\_name: mongodb

ports:

- "27017:27017"

networks:

- backend\_network

rabbitmq:

image: rabbitmq:3-management

container\_name: rabbitmq

ports:

- "5672:5672"

- "15672:15672"

networks:

- backend\_network

prometheus:

image: prom/prometheus

container\_name: prometheus

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml

ports:

- "9090:9090"

networks:

- backend\_network

grafana:

image: grafana/grafana

container\_name: grafana

ports:

- "3001:3000"

depends\_on:

- prometheus

networks:

- backend\_network

networks:

backend\_network:

driver: bridge

1. **Prometheus Configuration**

**prometheus.yml**:

yaml

Copy code

global:

scrape\_interval: 15s

scrape\_configs:

- job\_name: 'node\_backend'

static\_configs:

- targets: ['backend:3000']

1. **Running the Application**

To build and start all services:

bash

Copy code

docker-compose up --build

**4. Testing**

**Running Tests**

1. **Unit Tests**

Use Mocha and Chai to run unit tests for authentication and queue management.

**Run Tests**:

bash

Copy code

npm run test

1. **Interpreting Test Results**

* **Passed Tests**: Indicates the functionality is working as expected.
* **Failed Tests**: Review the error messages to identify and fix issues in authentication, request handling, or queue management.

**Sample Test Results**

If the tests pass successfully, you should see output like:

Authentication

✓ should register a new user

✓ should login a user

Queue Management

✓ should enqueue a request

3 passing (10ms)

**Failed Tests** will show details about what went wrong, which helps in debugging and improving the code.