**Intelligent Load Balancer for**

**Multi-API Management**

This documentation outlines the design, implementation, and usage of an Intelligent Load Balancer for managing and distributing traffic across multiple API endpoints. The load balancer is designed to handle different API types (REST, GraphQL, gRPC) efficiently using dynamic routing and queue management strategies.

**Key Components**

1. **Dynamic Routing**: Routes incoming requests to API endpoints based on various criteria such as load, response time, and endpoint availability, api-type,
2. **Queue Management**: Manages incoming requests using different queuing strategies (FIFO, priority-based, round-robin) to optimize performance and load distribution.
3. **Logging and Metrics**: Winston library used Captures detailed metrics including request times, endpoint selection, and response times for analysis and monitoring.
4. **Port Management**: Runs the load balancer and mock APIs on different ports to simulate like a real-world environment.

**Tech Stack**

* **Node.js**: Backend server environment.
* **Express.js**: Web framework for Node.js used for API routing.
* **MongoDB**: NoSQL database for logging and metrics storage.
* **Postman**: API development and testing tool.
* **GitHub**: Version control and repository hosting.
* **Video Explanation**: Screencast explaining code implementation and functionality.

**API Types**

1. **REST (Representational State Transfer)**:
   * A widely-used architectural style for designing networked applications.
   * Uses standard HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.
   * Stateless communication between client and server.
2. **GraphQL**:
   * A query language for APIs that allows clients to request only the data they need.
   * Single endpoint for flexible data retrieval.
   * Strongly typed schema for defining data models.
3. **gRPC (Remote Procedure Call)**:
   * Modern, high-performance RPC framework.
   * Uses protocol buffers for efficient serialization.
   * Bi-directional streaming and advanced features like authentication and load balancing.

### Project Setup

#### Prerequisites

* Node.js (version 4 or higher)
* MongoDB (version 4 or higher)
* npm (Node Package Manager)

#### Installation

1. **Clone the Repository:**

git clone <https://github.com/NITJManish/manish-sharma/tree/main/wasserstoff/BackendTask>

cd load-balancer

1. **Install Dependencies:**

npm install

1. **Configure Environment Variables:** I have attached the .env file in the every server directory
2. **Start MongoDB:** Ensure MongoDB is running on your local machine or update the DB\_LOCAL\_URI in the .env file to point to your MongoDB server.
3. **Start All server first: Rest API server, GraphQL API server, gRPC API server.**
4. Start the Load Balancer: Go in the integrated terminal and type – npm run dev (same comand for all server)
5. **Testing API Endpoints:** Open postman

And use the attached my Postman Collection Link: <https://universal-escape-204491.postman.co/workspace/My-Workspace~ed90198a-748e-4396-8651-fdeab80ee4b1/collection/11282934-62b7ac1c-1002-4b7b-9e2d-d1879ba90f25>

**GitHub Repository**

* **Repository Link**: <https://github.com/NITJManish/manish-sharma>.

**Code Explanation Video**

* **Video Link**:
* The video provides a detailed walkthrough of the code implementation, highlighting key features, and demonstrating the functionality of the load balancer.

**Notes on Load Balancer**

* **Purpose**: Distributes incoming network or application traffic across multiple servers to ensure no single server is overwhelmed, optimizing resource utilization and ensuring high availability and performance.
* **Benefits**:
  + **Scalability**: Easily scales with increasing traffic and workload.
  + **Fault Tolerance**: Reduces the risk of server failures impacting service availability.
  + **Efficiency**: Balances load efficiently across servers based on predefined algorithms and criteria.

**Challenges and Solutions**

One of the main challenges was managing

high traffic volumes without compromising performance.

We addressed this by find all resources, studies and optimizing our routing algorithms and implementing an efficient queue management system.

**Thank You !**