**Backend Go Lang Engineer Assessment**

**Problem 1: Distributed Key-Value Store**

Design and implement a distributed key-value store system using Go Lang. The system should consist of multiple server nodes that communicate with each other to store and retrieve key-value pairs. The key requirements for the system are as follows:
1. **Scalability**: The system should be horizontally scalable, allowing the addition of new nodes to handle increased load.
2. **Consistency**: Ensure strong consistency guarantees across all nodes in the system, even in the presence of failures and network partitions.
3. **Fault Tolerance:** Implement mechanisms to handle node failures gracefully and maintain data availability and integrity.
4. **Concurrency:** Support concurrent read and write operations efficiently while maintaining data consistency.
5. **Partitioning**: Implement data partitioning strategies to distribute key-value pairs evenly across multiple nodes.

Your solution should include the following components:
- A Go Lang implementation of the server nodes capable of communication and coordination.
- Data structures and algorithms for distributed data storage and retrieval.
- Mechanisms for handling node failures, network partitions, and data replication.
- Implementation of concurrency control mechanisms to ensure data consistency and integrity.
- Detailed documentation explaining the design decisions, system architecture, and deployment instructions.

# (OR)

**Problem 2: Rate Limiting Middleware**
Develop a rate limiting middleware for a high-traffic web service written in Go Lang. The middleware should restrict the number of requests processed per client IP address within a specified time window to prevent abuse and ensure fair resource allocation. The key requirements for the middleware are as follows:
1. **Dynamic Configuration**: Allow dynamic configuration of rate limits for different endpoints and client IP addresses.
2. **Efficiency**: Implement efficient data structures and algorithms to track request rates and enforce rate limits with minimal overhead.
3. **Concurrency Safety**: Ensure concurrency safety to handle concurrent requests from multiple clients simultaneously.
4. **Expiry Mechanism**: Implement a mechanism to expire outdated rate limit records and reclaim resources efficiently.

Your solution should include the following components:
- A Go Lang middleware function to intercept incoming requests and enforce rate limits.

- Data structures and algorithms to track request rates and enforce rate limits.
- Configuration options for specifying rate limits for different endpoints and client IP addresses.
- Implementation of concurrency-safe mechanisms for tracking request rates and enforcing rate limits.
- Thorough testing to validate the correctness and performance of the rate limiting middleware.

**NOTE - Please upload all the links on this google form https://forms.gle/ZdU1bbztmY8F7WjbA**