Task Management Application Design Document

1. Designing the Application Architecture

1.1 Architecture Overview

The Task Management Application, aimed at enhancing task tracking and collaboration across teams, will be built on a microservices architecture to ensure scalability, flexibility, and maintainability. The frontend will be developed using React for a dynamic and responsive user interface, communicating with backend services through RESTful APIs. The backend services will be implemented in Node.js, running within Docker containers for easy deployment and scaling. Data persistence will be managed by MySQL, chosen for its reliability and support for complex queries.

1.2 Raft Consensus Algorithm Roles

- ***Leader*:** Manages all write operations (CREATE, UPDATE, DELETE) on the task data, ensuring data consistency by replicating logs to followers. The leader also handles heartbeat messages to maintain authority within the cluster.

- ***Follower*:** Stores replicas of the leader's log and participates in elections. Followers may serve read-only queries if they have up-to-date information, reducing the load on the leader.

- ***Candidate*:**.A node becomes a candidate in the case of a leader failure, initiating an election process among the nodes to elect a new leader

1.3 MySQL Schema Design

- ***Tasks Table*:** Stores task details, including `task_id`, `title`, `description`, `status`, `priority`, `created_at`, and `updated_at`. Primary key on `task_id`.

- ***Users Table*:** Contains user information such as `user_id`, `name`, `email`, and `password_hash`. Primary key on `user_id`.

- ***Assignments Table*:** Maps tasks to users with columns `assignment_id`, `task_id`, `user_id`, and `assigned_at`. Primary keys on `assignment_id`, foreign keys on `task_id`, and `user_id`.

- ***Indexing*:** Indexes on frequently queried columns like `status` and `priority` in the Tasks table, and `email` in the Users table to speed up search operations.

2. Implementing Raft Consensus Algorithm

2.1 Raft Implementation

The Raft consensus algorithm will be implemented in the backend services using a Raft library tailored for Node.js, ensuring a modular and maintainable codebase. This library will manage the state machine, log replication, and leader election processes integral to Raft, ensuring consistency across the distributed system.

2.2 Communication Protocols

- ***Heartbeat Mechanism*:**Implemented via periodic AJAX calls from the leader to followers, ensuring continuous monitoring and leader authority.

- ***Log Replication Protocol*:** Utilizes HTTP POST requests for the leader to replicate logs to followers, with JSON payloads containing log entries.

- ***Election Process*:** Involves a combination of HTTP GET requests for voting and POST requests for announcing election results, with timeouts managed through Node.js's asynchronous capabilities.

2.3 Testing and Validation

- ***Scenario Testing*:**Will be conducted using a combination of unit tests (for individual module testing) and integration tests (for testing inter-module interactions and API endpoints), focusing on scenarios like leader failure, network partition, and data consistency validation.

- ***Performance Metrics*:** Response times, throughput, and fault recovery times will be measured under various loads using tools like JMeter and Prometheus, ensuring the system meets performance benchmarks.

.