## Homework 1: Distributed Key-Value Store with Fault Tolerance

**Objective**: Design and implement a distributed key-value store with fault tolerance using replication.

Requirements:

1. Replication Factor:
   1. Implement replication across multiple servers.
   2. Allow users to specify a replication factor (e.g., replicate each key-value pair across 3 servers).
2. Write DB:
   1. Manage writes toward a write DB.
   2. Allow reads from any replica.
3. Fault Tolerance:
   1. Handle server failures.
   2. Demonstrate the system's behavior when a replica fails and recovers.

Challenging Trade-offs:

1. Balancing consistency with availability.
2. Write to read protocol impacts on performance and availability.

Experiments:

1. Test the system's fault tolerance by simulating server failures and recoveries.
2. Measure the performance differences between at least two replication strategies.

Submission:

1. Implementation in Python.
2. Design document with system architecture, consistency model choice, and fault tolerance strategies.
3. Report with experimental results and analysis.

## Homework 2: Real-Time Collaboration Platform with Network Partition Tolerance

**Objective**: Design and implement a real-time collaboration platform that can tolerate network partitions.

Requirements:

1. Shared Document Editing:
   1. Implement a collaborative editor that allows multiple users to edit the same document in real time.
   2. Use WebSockets for real-time communication.
2. Consistency Model:
   1. Implement a consistency model that allows updates to be eventually consistent after network partitions.
   2. Resolve conflicts using a strategy (study: Operational Transformation (OT) or Conflict-free Replicated Data Types (CRDTs)).
3. Network Partition Tolerance:
   1. Simulate network partitions.
   2. Ensure that all edits are merged after the partition is resolved.
4. User Presence Notification:
   1. Notify users when someone joins or leaves the document editing session.

Challenging Trade-offs:

1. Consistency vs. availability during network partitions.
2. Conflict resolution strategies in collaborative editing.

Experiments:

1. Simulate network partitions and measure the system's ability to reconcile changes.
2. Analyze the impact of conflict resolution strategies on consistency and latency.

Submission:

1. Implementation in Python/JavaScript.
2. Design document describing the architecture, conflict resolution strategies, and partition tolerance mechanisms.
3. Report with experimental results and analysis.

## Homework 3: Distributed Load Balancer with Consistent Hashing

**Objective**: Design and implement a distributed load balancer using consistent hashing to manage traffic distribution.

Requirements:

1. Load Balancer Implementation:
   1. Implement a load balancer that distributes incoming requests among a set of backend servers.
   2. Use consistent hashing to map requests to servers.
2. Server Health Check:
   1. Implement a mechanism to check the health of backend servers periodically.
   2. Remove unhealthy servers from the consistent hashing ring.
3. Dynamic Scaling:
   1. Add or remove servers dynamically, ensuring minimal data movement due to consistent hashing.
4. Client Library:
   1. Provide a client library that abstracts the load balancer’s details and allows easy integration.

Challenging Trade-offs:

1. Achieving load balancing with minimal disruption when servers are added or removed.
2. Managing consistent hashing with dynamic scaling and fault tolerance.

Experiments:

1. Measure the distribution of traffic across servers after adding or removing servers dynamically.
2. Simulate server failures and measure the effect on traffic distribution.

Submission:

1. Implementation in Python.
2. Design document describing the consistent hashing strategy, health checks, and dynamic scaling mechanism.
3. Report with experimental results and analysis.

## Homework 4: Designing an NFT-Based Fight Game

In this homework, you will design a blockchain-based game called "Fight Club" that uses Non-Fungible Tokens (NFTs) to represent unique characters and a utility token for in-game transactions and mechanics. The game will involve characters fighting each other, with outcomes determined by the characters' properties and randomness.

**Objectives**:

Design the NFT Character System:

* Create a smart contract for NFT characters with random properties.
* Properties should influence the outcome of fights (e.g., strength, agility, intelligence).

Design the Utility Token System:

* Create a utility token (ERC-20 standard) for in-game transactions.
* Implement mechanics to earn and spend tokens.

Develop Fight Mechanics:

* Implement fight logic using the properties of NFT characters.
* Ensure randomness plays a role in the fight outcomes.

Integrate Frontend with Smart Contracts:

* Create a simple web interface for minting characters, initiating fights, and viewing results.
* Use web3.js or ethers.js to interact with the blockchain.

**Submission Requirements:**

* Smart Contracts: Submit the Solidity code for the NFT, utility token, and game mechanics.
* Frontend: Submit the HTML and JavaScript code for the web interface.
* Report: Include a report explaining the design choices, how the contracts interact, and any challenges faced.

**Grading Criteria:**

* Functionality: Does the game work as expected?
* Code Quality: Is the code well-organized and commented?
* Innovation: Are there any additional features or improvements made?
* Report: Is the report comprehensive and well-written?