**Team: Bhavesh, Prashanth**

**Architecture of the Distributed User Store Project:**

A client can send requests to a distributed user store consisting of multiple nodes according to the project architecture. Write operations are governed by a token-based mutual exclusion mechanism, in which only the token-holding node is allowed to write. To ensure data consistency and replication, the token is manually transferred to a different node after it has been written.

**Explanation of each file:**

**Distributed User Store:**

A basic distributed system consisting of a primary node and three backup nodes is established by the supplied Java code, which also defines the DistributedUserStore class. It links the nodes together and launches separate servers on various ports (8080–8083) for every node. The application gets the nodes ready to manage requests in a dispersed setting.

**Node.java:**

The Node class utilized in a distributed system is defined in the Node.java file. Every node has a user store, a list of neighbors, a token status, and an ID. Starting with a token, the primary node can replicate data to backup nodes, such as user information. It also makes it easier to transfer the token to a different node for regulated actions.

**Node server:**

For every node in the distributed system, a server thread is defined in the NodeServer.java file. It responds to client requests, executes commands "ADD" (to add or update a user) and "READ" (to retrieve user information), and manages write operations based on tokens. After every write operation, the token is manually transferred to the following node.

**User store clint:**

A straightforward client that connects to a given server (node) on a given port is defined in the UserStoreClient.java file. For commands like adding or reading user data from the distributed system, it requires input from the user. These commands are sent to the server by the client, which then prints the response to the user.

The system will start with four nodes, from which Node1 is the primary and Node2, Node3, and Node4 are the backup nodes. Each of them listens on a different port, starting from 8080 to 8083. Node1 has the token, runs the write requests, and propagates the change to the backup nodes. Finally, Node1 manually transfers the token at the end of each operation to Node2. Clients will use UserStoreClient.java for connecting with the nodes with ADD and READ commands. This would ensure strict consistency, whereby an update-for example, adding a user is replicated by every node, and it can read updated data from any node, hence validating successful replication and token-based mutual exclusion.

**Out puts:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

We also implemented it in VM.

Bhavesh:

Configured primary and backup nodes in the system. Added neighbours to the primary and backup nodes so that they could talk to each other.

Implemented server-side logic to handle the client request using commands like "ADD" for insert/update and "READ."

Ensured strict consistency by having all updates from the primary node copied to all the backup nodes. Provided the replication method such that the moment any user data gets updated at the primary node, it got replicated.   
Implemented token-based mutual exclusion at the server side, whereby only the node possessing the token was granted permission to write. A node not in possession of the token should not perform any write operation. The node correctly handled the lifecycle of the token.Involed in testing and debugging.

Prashanth:

Added the Node class with attributes like nodeId, isPrimary, neighbors, and a user store.

Methods to manage the relationship between the primary and the backup nodes were provided, including adding neighbors.

In case of an ADD operation, the system updates the primary node and, by default, will trigger replication to maintain strict consistency in all the backups.

Implemented the passToken method for safe token passing from one node to another.

Followed mutual exclusion by keeping in coordination with token management in the server class. Involved in testing and debugging.

**Challenges faced**:

While performing token-based mutual exclusion: out of all nodes, only the node possessing the token was allowed to perform a write operation. Manual management of token passing between nodes was quite cumbersome, and it was challenging to avoid losses or duplication of the token during transitions.

As updates were replicated from the primary to backup nodes, it was very difficult to maintain strict consistency across the nodes. This always required careful error handling, as inconsistencies between nodes in real time usually resulted under load conditions.Handling node failures, especially on the primary token-holding node.