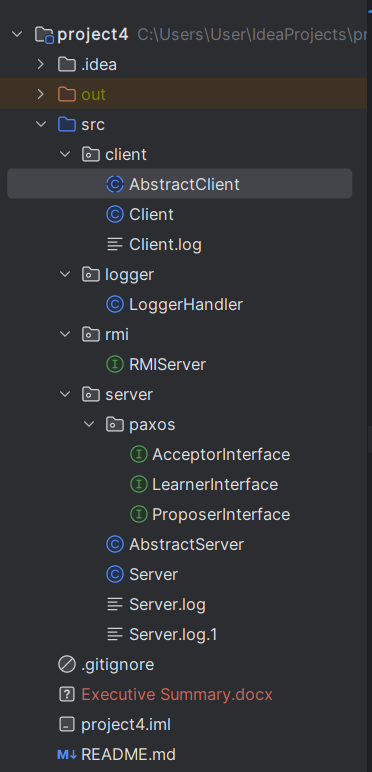
**ASSIGNMENT OVERVIEW:**

**Paxos Implementation:**

* Integrate Paxos roles, including Proposers, Acceptors, and Learners, into your system.
* Focus on the algorithmic steps involved in realizing consensus in event ordering.
* Allow client threads to generate requests to any replica at any time.
* Optionally, implement leader election among proposers to minimize the potential for live lock.

**Random Failures:**

* Configure acceptors to "fail" at random intervals to simulate server failures.
* Implement acceptor threads to periodically fail, possibly using a timeout mechanism.
* After failure, restart the acceptor thread after a delay, resuming its functions with potential state differences.



**TECHNICAL IMPRESSION:**

In the context of distributed databases, the Paxos consensus algorithm is adapted to manage a sequence of operations that collectively define the database's state. This sequence is upheld through a "Log," where each entry corresponds to an operation, and maintaining the order of these operations is crucial for achieving a consistent database state.

Rather than focusing on agreeing on a single value, the Paxos implementation is tailored to reach consensus on which operation should be executed at a specific position within the log. The log begins with position 0, and each proposer is responsible for proposing an operation for a given log position. Upon acceptance of a proposal, the log position counter is incremented by 1, ensuring a sequential and consistent evolution of the database state. This approach ensures that if the same sequence of operations is applied, the distributed databases will converge to an identical state.

In this project, prepare, accept, learn and propose methods in the Server code are essential components of the Paxos consensus algorithm. These methods are part of the roles played by a server in the Paxos protocol: Proposer, Acceptor, and Learner.

**Prepare Method (prepare):**

* The prepare method is invoked by a Paxos proposer to initiate the consensus process.
* It takes a proposal ID (proposalId) as a parameter, which is a unique identifier for a particular proposal.
* The server compares the received proposalId with its current sequence number (currentSequenceNumber).
* If the received proposalId is greater than the server's current sequence number, the server promises not to accept any proposals with a lower ID.
* The server sets its isPromised flag to true, indicating that it has promised not to accept lower-numbered proposals.
* The method returns 1 to indicate a successful promise or 0 if the promise is not granted.

**Accept Method (accept):**

* The accept method is invoked by a Paxos proposer after receiving promises from a majority of acceptors.
* It takes the proposal ID (proposalId) and the proposed value (proposalValue) as parameters.
* The proposer sends the accept message to all acceptors, instructing them to accept the proposal.
* In the provided code, the accept method is implemented to loop through all acceptors and call their learn method, passing the proposal ID and value.

**Propose Method (propose):**

* The propose method is invoked by a Paxos proposer to initiate the Paxos consensus process.
* It takes a proposal ID (proposalId) and a proposed value (proposalValue) as parameters.
* The proposer sends a prepare message to all other acceptors to request their promises not to accept proposals with a lower ID.
* After receiving promises from a majority of acceptors, the proposer sends an accept message with the proposal ID and value to instruct the acceptors to accept the proposal.

**Learn Method (learn)**

* this.setProposalValue(acceptedValue) is used to store the accepted value locally. The accepted value typically represents an operation that has achieved consensus through the Paxos protocol.
* The method then calls this.acceptRequest(...), passing information extracted from the accepted value.
* Type casting is performed on acceptedValue to the type Operation, indicating that the accepted value is expected to be an instance of the Operation class.
* Extracts details such as clientMessage, serverResponse, clientAddress, and serverId from the accepted operation.
* Invokes the acceptRequest method, that will further process and execute the operation on the server, ensuring that the server's state reflects the agreed-upon sequence of operations in the Paxos consensus.

These methods together form the core of the Paxos consensus algorithm, allowing a group of servers to agree on a single value even in the presence of failures and network partitions. The proposer initiates the process, the acceptor promises not to accept conflicting proposals, and the learner ultimately learns the agreed-upon value.

**PURPOSE AND SCOPE OF THE PROJECT:**

The purpose of this project is to enhance the fault tolerance and consistency of a replicated Key-Value Store Server by implementing the Paxos consensus algorithm. The key objectives and purposes of the project include:

**Fault** **Tolerance**:

* Address the limitations of two-phase commit protocols used in the previous project by introducing fault tolerance mechanisms.
* Implement Paxos to achieve consensus among replicated servers, ensuring the system's continued operation despite failures of individual replicas.

**Consensus in a Distributed System:**

* Utilize Paxos to establish consensus on the order of events and updates in a distributed system with multiple replicas.
* Ensure that all replicas agree on the sequence of operations to maintain a consistent state across the replicated Key-Value Store.

**Algorithmic Understanding:**

* Gain a deep understanding of the Paxos consensus algorithm and its steps for reaching agreement among distributed components.
* Implement Paxos roles, including Proposers, Acceptors, and Learners, to orchestrate the consensus process.

**Client-Server Interaction:**

* Allow client threads to interact with any replica at any time, providing flexibility in client-server communication.
* Enable the system to handle concurrent client requests while maintaining consistency through Paxos.

**Random Failures Simulation:**

* Simulate random failures, particularly in acceptor threads, to mimic real-world scenarios where servers may fail intermittently.
* Demonstrate how Paxos overcomes replicated server failures by periodically restarting failed components.