Armazon - Class 8 Group 1

- Diogo Filipe Ferreira da Silva up202004288
- Henrique Oliveira Silva up202007242
- Tiago Nunes Moreira Branquinho up202005567

Planned Project Design

Frontend

While it may not be of primary importance in this project, we plan to develop a web page using the Python library Flask. Flask can handle incoming data from clients or devices and distribute it to the appropriate components of the distributed system. Users will have the ability to perform all necessary actions through the web page, including adding shopping lists, deleting shopping lists, adding items, and updating quantities. This addition to the project will facilitate testing and demonstration.

Database Structure

We plan to implement a key-value stores database using SQLite. That way, each shopping list id, representing a key generated with UUID4 will be associated to JSON data which includes the items present in that shopping list. This approach will eliminate the need for relational queries associated with structured relational databases. Altough those models allow for strong consistency on data, they also cause inefficiencies, limiting scaling and availability.

Server Structure

To satisfy the need of creating a local-first shopping list application that allows data sharing among users, it is intended to implement a decentralized distributed architeture. Data storage will be distributed across multiple nodes, which will be able to communicate among themselves to allow data replication, thus providing fault tolerance and reducing data access times. The replication will be asynchronous, meaning it is designed to be an eventually consistent data store; that is all updates reach all replicas eventually.

Each shopping list will be consistently hashed to one of the server nodes, ensuring that each request consistently accesses the same node. This approach also helps balance the load across the nodes. In the event of a failure, the system will detect it through timeouts and subsequently search for a node containing a replica.

Client Interaction

Clients will hold a local data storage, that will remain updated even in case of server failure and will allow to update the server data once it is back to availability. Clients should also be able to access the same shopping list concurrently and it should be “always writeable”, where no updates are rejected due to failures or concurrent writes.

Conflict Handling

At any point, if data discrepancies occur among any participants in the system (either between servers or between clients and servers), a merge process will be initiated. In the event of conflicts, the most recently

updated data will take precedence. Conflicts between causally unrelated data are resolved during read operations, ensuring that write operations are never rejected, to protect user experience.

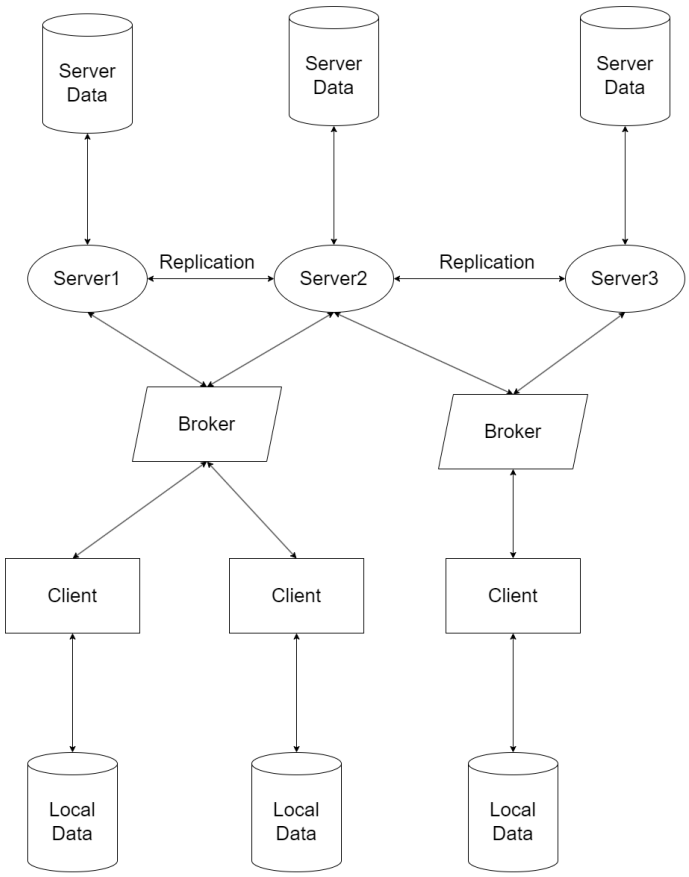
Methods

- get(key) and put(key, context, object), objects(shopping list data) are associated with a key(shopping list ID).
- get:
  - Locates object replicas associated with the provided key in the storage system.
  - Returns either a single object or a list of objects with conflicting versions, along with a context.
- put:
  - The placement of replicas are determined based on the key.
    - Utilizes MD5 hash to generate a 128-bit identifier from the key.
  - Writes the replicas to disk.
  - The context, which encodes system metadata about the object and its version, is stored along with the object. This enables the system to verify the validity of the context object.

CRDTs

Regarding the usage of CRDTs, due to the scalability requirements, we are focused on the DeltaCrdt, since it doesn't require transmission of its state with every change made, only of its delta. That logic will be implemented to the LWW-Element-Set CRDT.

Sistem Design Diagram



This is a reduced example of the design architeture. In a larger scale each shopping list should contain more replicas.