Shopping Lists on the cloud

Large Scale Distributed Systems - FEUP

Afonso Abreu - up202008552
João Malva - up 202006650

2023/2024

Introduction

The goal of this project wishes to achieve a local-first shopping list application. It enables users to manage numerous shopping lists, each one identified by its own ID. This way, it allows users to collaborate with each other as the data is shared between machines.

This type of system should achieve the following requirements:

01

Offline Access

A user is allowed to make changes to a shopping list even if there isn’t a connection to the servers.

02

Users collaboration

Many users can edit the same shopping lists granting concurrency to the system.

03

Data Sharing between clients

Since data is shared between users, consistency is key.

04

High Availability

We have to make sure our system works without bottlenecks even for a alrge group of users

System Architecture

- **Local First App:** Comes up with an user interface so that the clients can connect to the cloud. However the app can be used offline.
- **Proxy:** Estabilishes the connection between clients and the servers. Acts like an invisible facilitator esnuring data sharding and replication.
- **Main Node:** Acts like a pinpoint for other nodes to know where each one is located.
- **Node:** Receives requests related to the shopping lists.

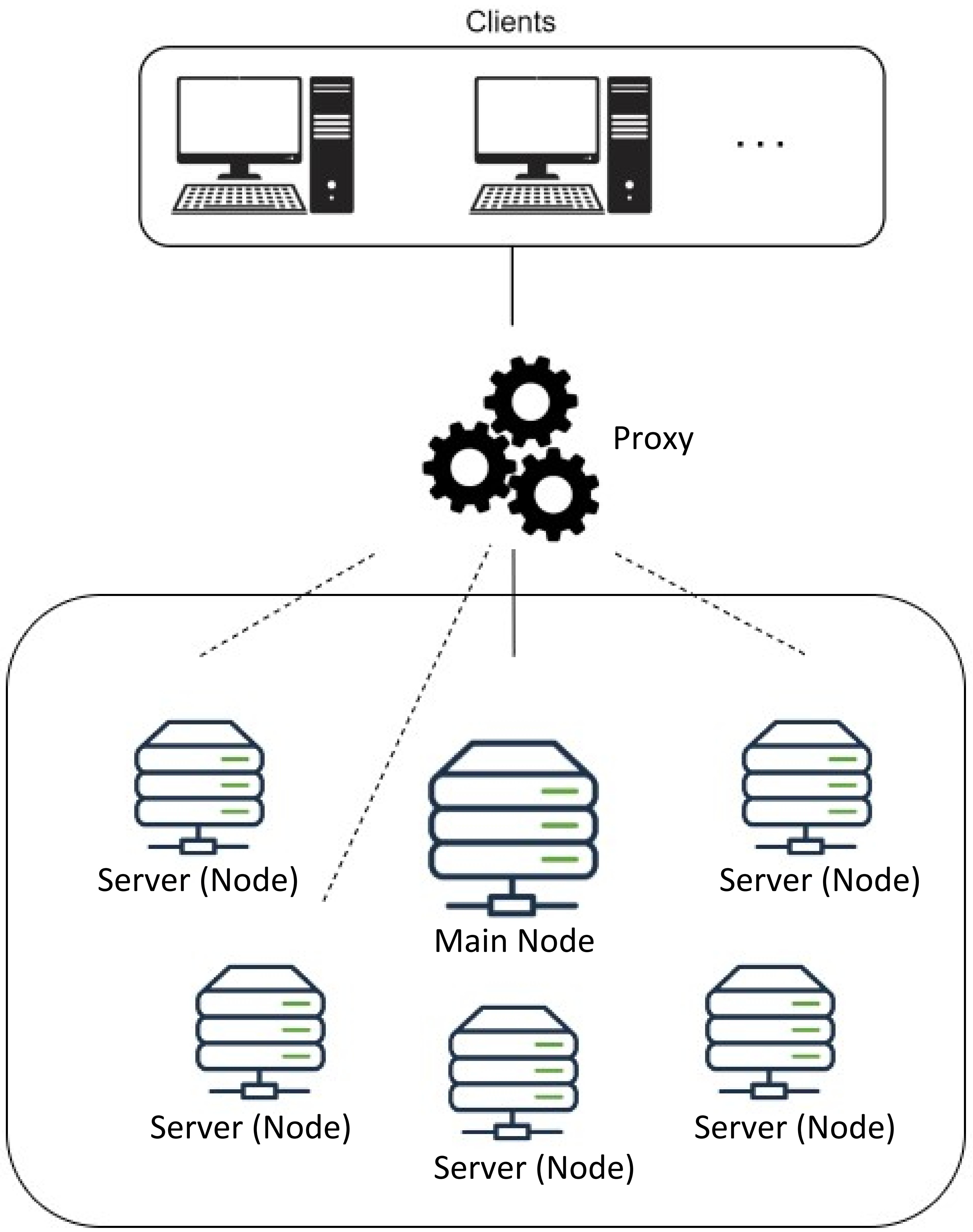


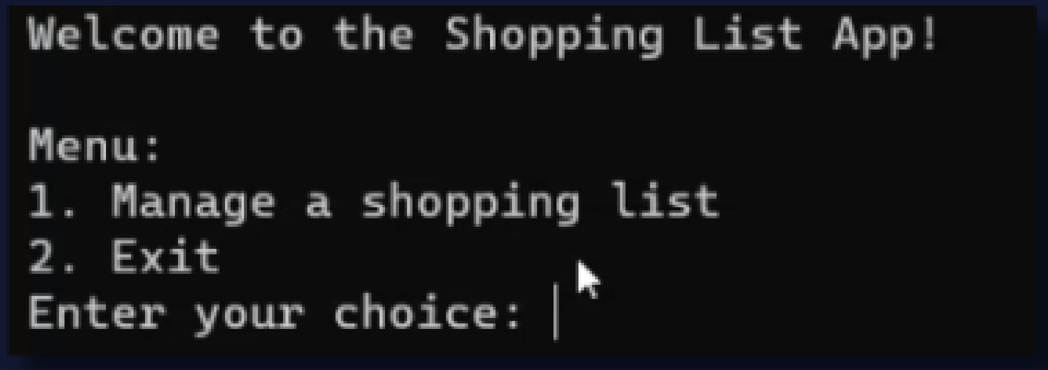
Fig 1: System Architecture

Implementation

Client Side

Our application uses a Local-First approach and so it prioritizes local data storage and processing. It provides a seamless user experience with or without an internet connection and because of that it gives control to the users over their data.

When the application starts, it is created a storage for shopping lists. The user then chooses wether he wants to manage or remove a shopping list, through its ID. If one wants to manage a shopping list that doesn’t exist, the system creates a new one with the ID used on search.



```
Welcome to the Shopping List App!

Menu:
1. Manage a shopping list
2. Exit
Enter your choice:
```

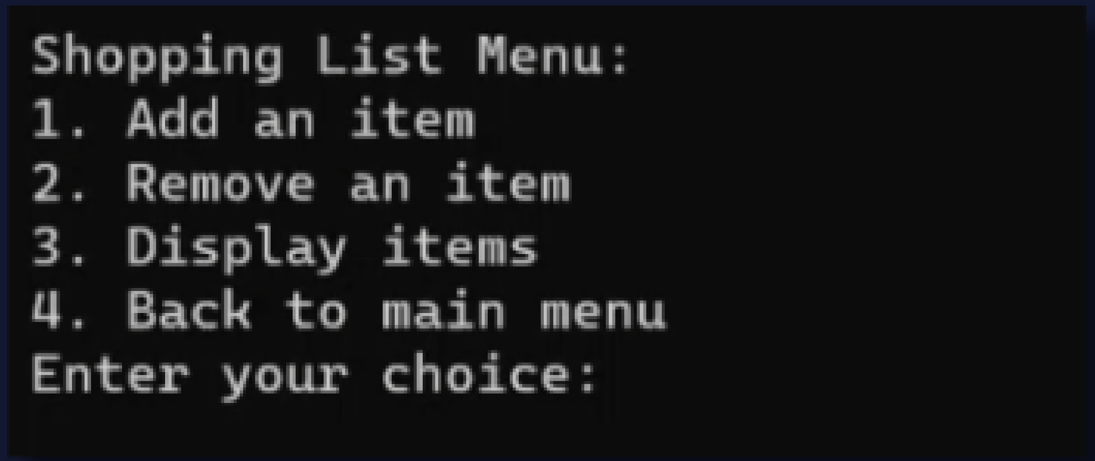
Fig 2: Main Menu

Implementation

Client Side

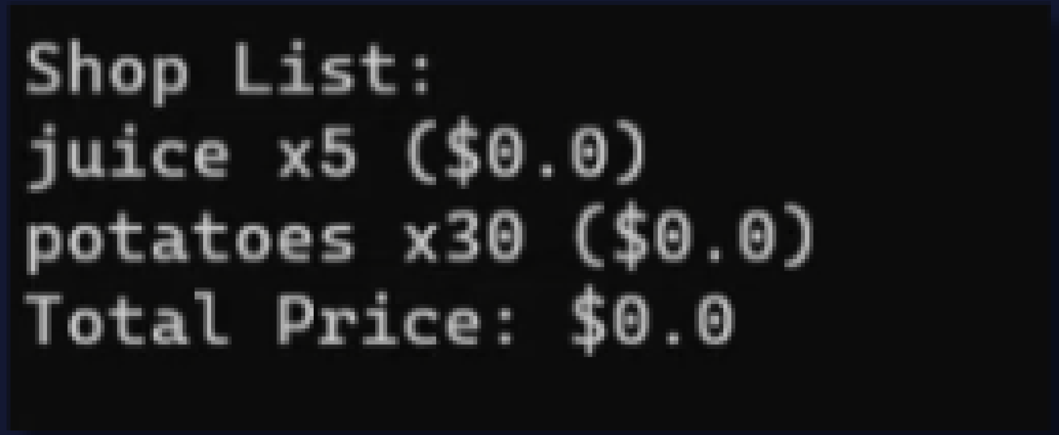
Once the user gets to manage a shopping list there are three options to choose: to add items, to remove items or to display the current items that belong to that shopping list.

An item has also a quantity number associated with it. that means that if that number reaches zero, the item simply is not displayed.



```
Shopping List Menu:
1. Add an item
2. Remove an item
3. Display items
4. Back to main menu
Enter your choice:
```

Fig 3: Shopping List Menu



```
Shop List:
juice x5 ($0.0)
potatoes x30 ($0.0)
Total Price: $0.0
```

Fig 4: Shopping List Display Menu

Implementation

Communication

Our communication method is based on TCP ensuring data consistency and coherence.
The messages are traded via ZeroMQ.

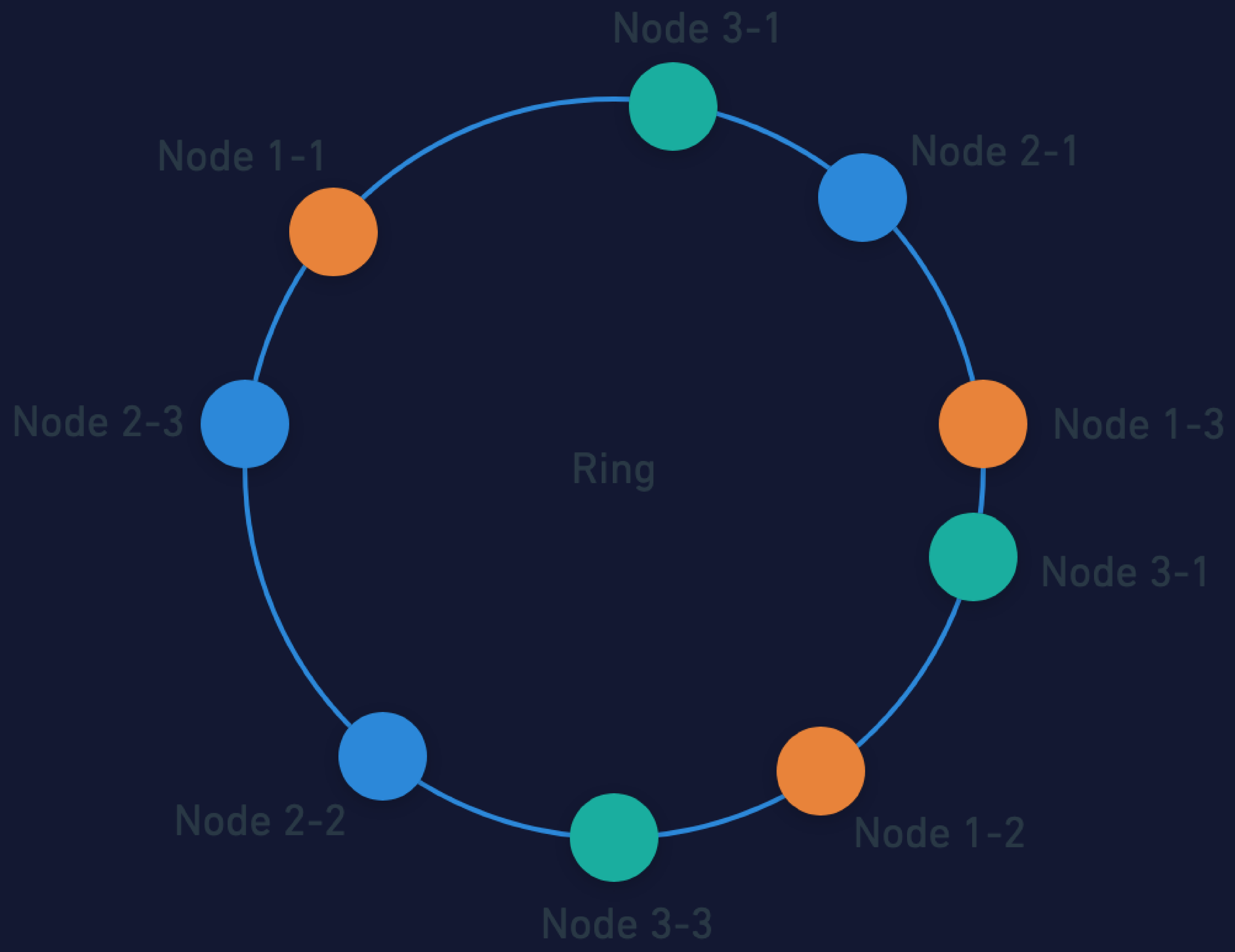
It exists communication between:

- Client and Proxy;
- Proxy and Nodes (Servers) in order to evenly distribute tasks across them;
- Main Node and Proxy so that the Nodes addresses are known;
- Main Node and Nodes (unidirectional) to update the active nodes list;

Implementation

Hash Ring

Our systems also takes into account Consistent Hashing. It operates by assigning the data objects and nodes a position (token) on a virtual ring structure (hash ring).
This way, once a new node is added to the ring the workload is immediatly divided through all nodes. At the same time, the Proxy list of nodes is also updated.



Implementation

Error Handling

- The error handling strategy adopts a defensive approach by validating inputs, setting timeouts to prevent blocking, and responding appropriately to unexpected or incorrect scenarios.
- The use of conditional checks and timeouts ensures that the system remains resilient and capable of recovering from potential issues without compromising the overall stability of the distributed key-value store proxy.

Implementation

Resolution of Conflicts

- This system allows users to manage shopping lists trough their ID.
- Means that one shopping list can be edited by many users at the same time, leaving us with the problems of data sharing and its resolution of conflicts.
- However, since we wanted a system with maximum efficiency we were tempted on using CRDT’s (Conflict-free replicated data types).
- By synergizing the G-Set's (grow-only) set management with LLW's (last-writer-wins) approach to item quantities, our distributed system attains a balance between scalability, consistency, and the dynamic management of item updates. This combined strategy lays the foundation for a resilient and adaptable architecture.

Conclusion

We achieved our goal of creating a Local First system capable of managing numerous shopping lists through their ID.
Data is shared between users and we took into account important topics such as scalability and availability which combined make our system more reliable.
Besides that, we adopted some error handling techniques in order to achieve a flexible network.
Some aspects we could improve:

- User interface. Right now we only have a terminal based interface;
- Bettet use of CRDT;
- Timeouts in certain operations;
- Incabability of managing the Nodes states while running;
- More complete Key-Value Database;