LARGE SCALE DISTRIBUTED SYSTEMS

SHOPPING LISTS ON THE CLOUD

Diogo Alves up202410346
Inês Almeida up202004513
Manuel Alves up201906910

CONTENTS

PROBLEM DESCRIPTION

CLIENT

DATABASE

ARCHITECTURE

CRDTs

ZMQ

MAIN CHALLENGES AND LIMITATIONS

CONCLUSION

PROBLEM DESCRIPTION

PROJECT OBJECTIVE

Development of a local-first shopping list application with data persistence on devices and in the cloud, using ZeroMQ for efficient communication, enabling collaboration and reliable backup

CORE OPERATIONS

Create and share unique shopping lists for collaborative editing

Manage items: add, remove, mark as acquired, or update quantities

USER EXPERIENCE GOALS

Real-time synchronization for seamless collaboration

High availability and data persistence

SCALABILITY AND RELIABILITY

Support concurrent updates from multiple users

Ensure consistency using CRDTs

Back to Contents

CLIENT

POSSIBLE OPERATIONS

Add or Increment Items

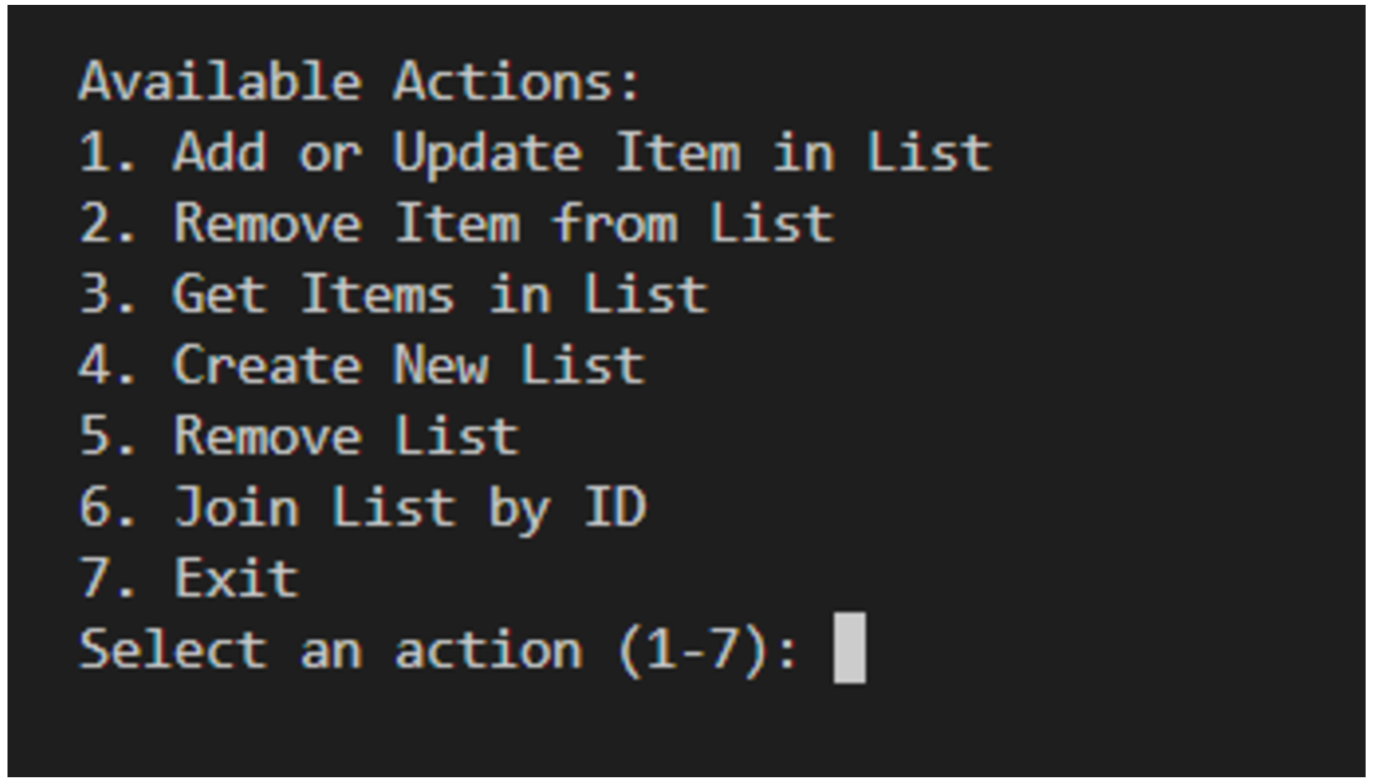
Remove Items

Get Items in a List

Create a New List

Remove a List

Join a List by ID



```
Available Actions:
1. Add or Update Item in List
2. Remove Item from List
3. Get Items in List
4. Create New List
5. Remove List
6. Join List by ID
7. Exit
Select an action (1-7): █
```

The client uses **polling** to periodically check for updates on the server and synchronize its local database

Back to Contents

DATABASE

Each server and client maintains its own JSON file as a database

DATABASE STRUCTURE

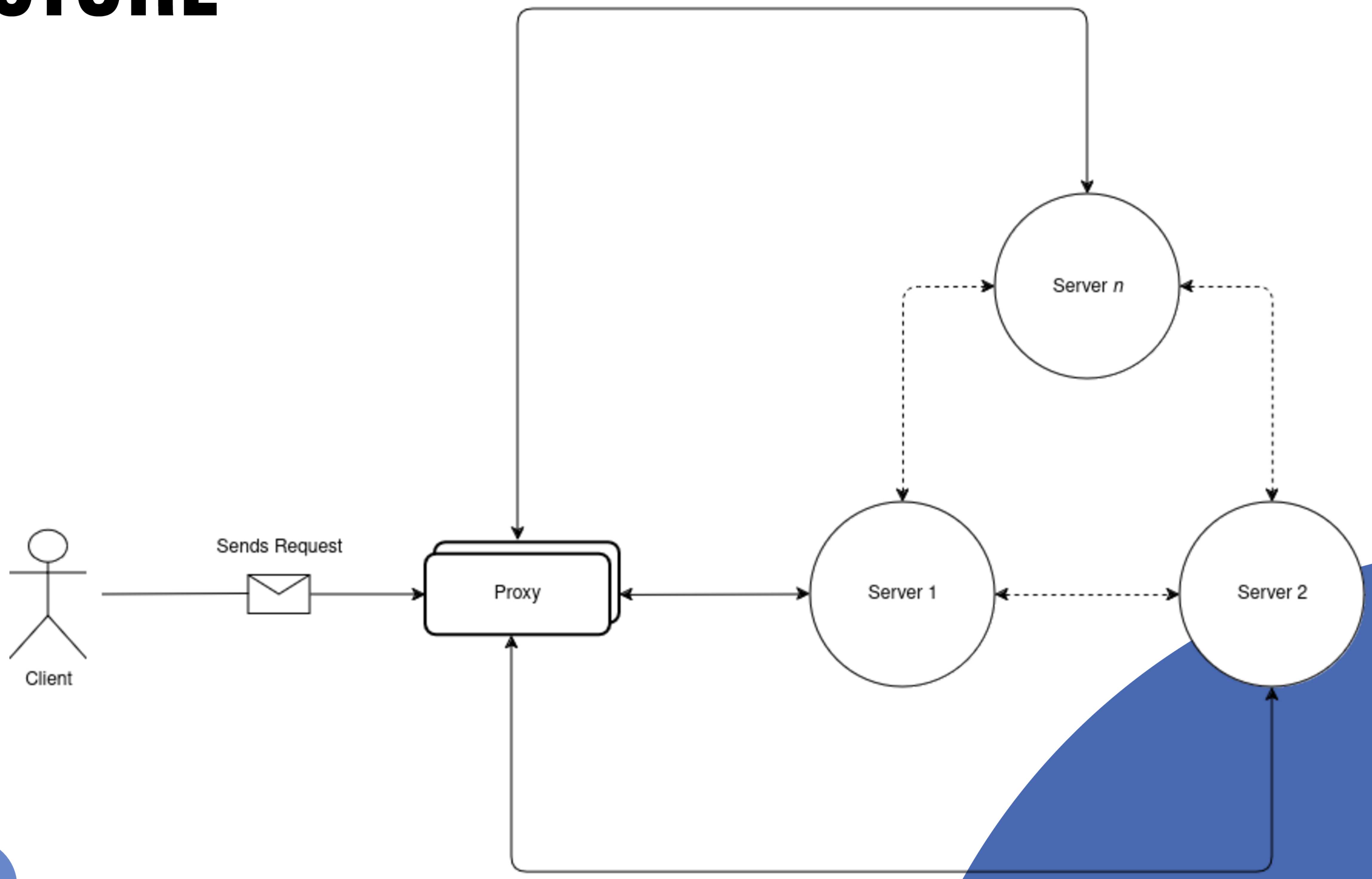
lists

Each list has:

- **id**: A unique identifier for the list
- **name**: A user-friendly name for the list
- **items**: A collection of objects, each with:
  - **Item**: the name of the specific item
  - **Quantity**: a field that tracks the number of items

Back to Contents

ARCHITECTURE



Back to Contents

Back to Contents

CRDTs

OR-SET (OBSERVED-REMOVE SET)

WHAT IT IS?

A CRDT that tracks adds and removes independently to resolve conflicts in distributed systems

HOW IT WORKS?

Step 1: Track adds and removes in separate sets

```
add_set
remove_set
```

Step 2: Resolve Conflicts in Item Existence

An item remains in the list if its additions outweigh its removals

```
self.add_set[element]['Quantity'] -
self.remove_set[element]['Quantity'] > 0
```

LWW-SET (LAST-WRITE-WINS SET)

WHAT IT IS?

A CRDT that uses timestamps to resolve conflicts, ensuring the most recent operation (**add** or **remove**) prevails

HOW IT WORKS?

Step 1: Track timestamps for operations

Assign timestamps to each set

Step 2: Resolve conflicts in Item Quantity

Compare timestamps for each item's operations, and the one with the latest timestamp wins

```
if current_timestamp <= timestamp:
    self.add_set[element[0]]["timestamp"] = timestamp
    self.add_set[element[0]]["Quantity"] = element[1]
```

The implemented CRDTs successfully ensure conflict resolution, consistency, and reliability in managing item states and quantities in our project

Back to Contents

ZMQ

A high-performance messaging library for distributed systems that enables efficient inter-process communication using various patterns

COMMUNICATION FLOW

Client → Broker	Sends request using **REQ** socket
Broker Routes Message	Uses **ROUTER/DEALER** pattern to forward the request
Broker → Server	Server receives request via **REP** socket
Server Processes Request	Executes logic and prepares response
Server → Broker	Sends response back to broker
Broker → Client	Client receives response via **REQ** socket

The broker intermediates clients and servers, enabling decentralized, scalable, and efficient communication by routing requests, balancing load, and decoupling interactions

MAIN CHALLENGES AND LIMITATIONS

MAIN CHALLENGES

- Integration of the CRDTS fully working, assuring consistency in both client and server databases.

LIMITATIONS

- Only works properly with one server (couldn’t integrate the hashing properly, in order to have consistency between servers)
- Console as the interface.

Back to Contents

Conclusion

The project successfully achieved its goal of creating a collaborative and scalable shopping list app, utilizing ZeroMQ for decentralized communication and CRDTs for real-time consistency. Throughout its development, we deepened our understanding of distributed systems, efficient communication, and conflict-free data synchronization.

Back to Contents