SHOPPING LIST CLOUD APPLICATION

Diogo Fonte, up202004175
Domingos Santos, up201906680
Duarte Lopes, up202006408
Pedro Ramos, up202005460

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

INDEX

Problem Description and Requirements

Project context and requirements for the application.

Technical Solution

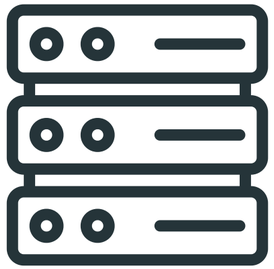
Main components description (Client, Cloud and Router/Load Balancer) and CRDT approach.

Solution Evaluation / Reflection

Identification of the limitations of the developed solution.

PROBLEM DESCRIPTION

Local-First Approach



Users can share their lists for collaborative editing via cloud

Development of a shopping list sharing application

Users can create and edit their lists locally, and backup in the cloud

REQUIREMENTS

Local First Approach

- Operating locally on the user’s computer.
- Be able to operate on lists, even if offline.
- Synchronization with the cloud when the connection is available.

Cloud

- Use of sharding to decompose the Cloud in several servers.
- Load distribution across available servers
- Replication of the lists in the several servers.

Share Shopping Lists

- Share the list code, so that other people can collaborate concurrently.
- Edit and update lists with other people's changes.

Conflict-Free Replication Data Types

- Implementation of a CRDT to deal with concurrent changes.
- Deal with increments/decrements of quantities and additions/removals

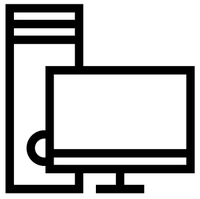
User Interface

- Ability to operate on shopping lists on the browser, with a basic graphical user interface.

TECHNICAL SOLUTION

CLIENT

- Runs on a port chosen at launch
- Continuously tries to connect to Router, if it is not available
- Creates and edits local lists
- Responsible for all the system interaction
- Communicates with the Router to perform Cloud-related client requests

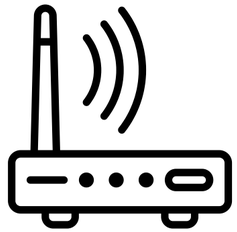


SERVER

- Runs on a port passed by argument at launch
- When connecting to Router, sends the current number of lists stored, and their codes
- If it receives a list code, checks if the corresponding list exists and forwards it to the Router if found
- If it receives a list from the Router, rewrites if it already exists in the cloud or creates a new one

ROUTER / LOAD BALANCER

- Forwards messages between Servers and Clients
- Has two Maps:
  - 1) elements: key = listCode and value = { } of servers with a replica of that list
  - 2) elements: key = serverCode and value = [serverConnection, numberOfLists]
- When it needs to backup a new list, searches for the two servers with the least amount of lists stored



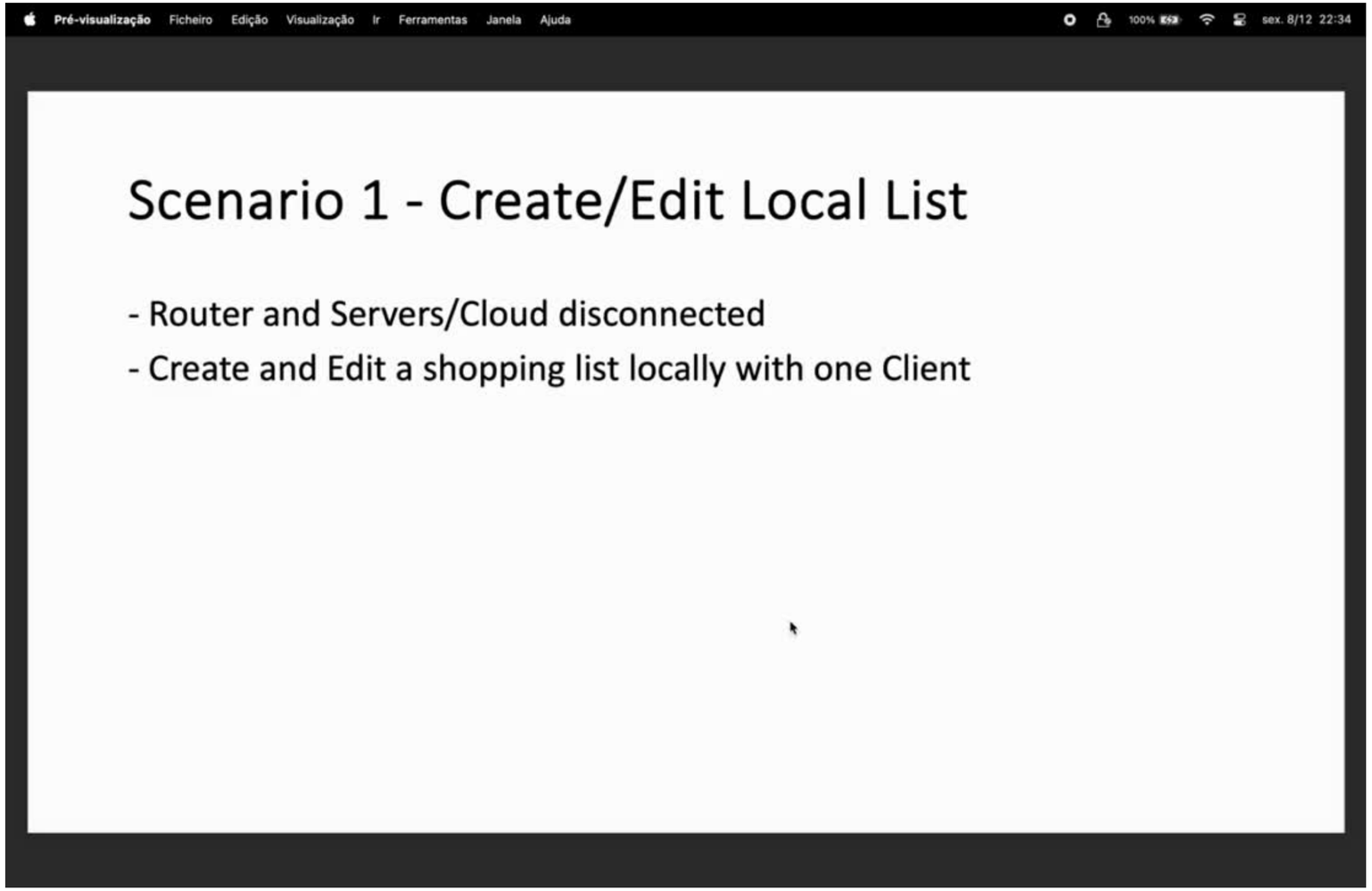
CRDT

- Use of PNCounter to the desired quantity
- Map of elements <listCode<string>, {addCounter<int>, removeCounter<int>}>
- Merge function that compares the additions and removals counters
- Add item operation increments the respective addCounter
- Remove item operation increments the respective removeCounter

PROJECT LIMITATIONS

- Only one Router available, but it is easily scalable
- When the Router shuts down after valid connections, we need to restart the Servers and the Clients
- Synchronization like git with merge actions, instead of automatic synchronization
- Only stores two replicas of each list in the Cloud, at the moment

DEMO VIDEO



link: https://github.com/JozeRamos/SDLE/blob/main/docs/demo/demo.mp4