The new shopping experience

ShopSync

Shopping lists on the cloud

SDLE – 2023/2024

Dinis Sousa - 202006303

João Matos - up202006280

João Pinheiro - up202008133

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Daniel Tinoco (Practical classes)

Carlos Baquero-Moreno (Theoretical classes)

Pedro Souto (Theoretical classes)

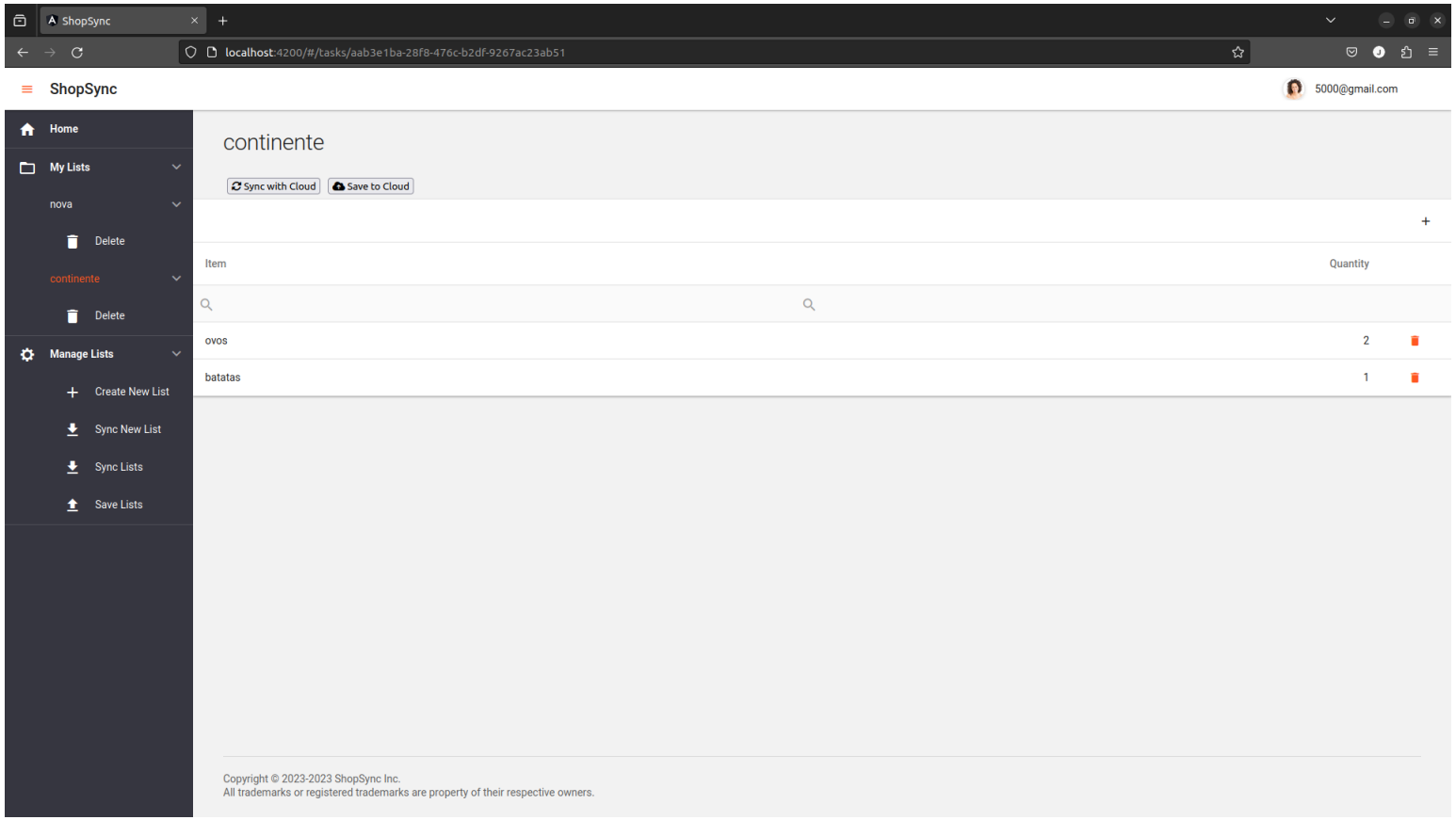
ShopSync is a local-first application that enables users to create, edit and share shopping lists

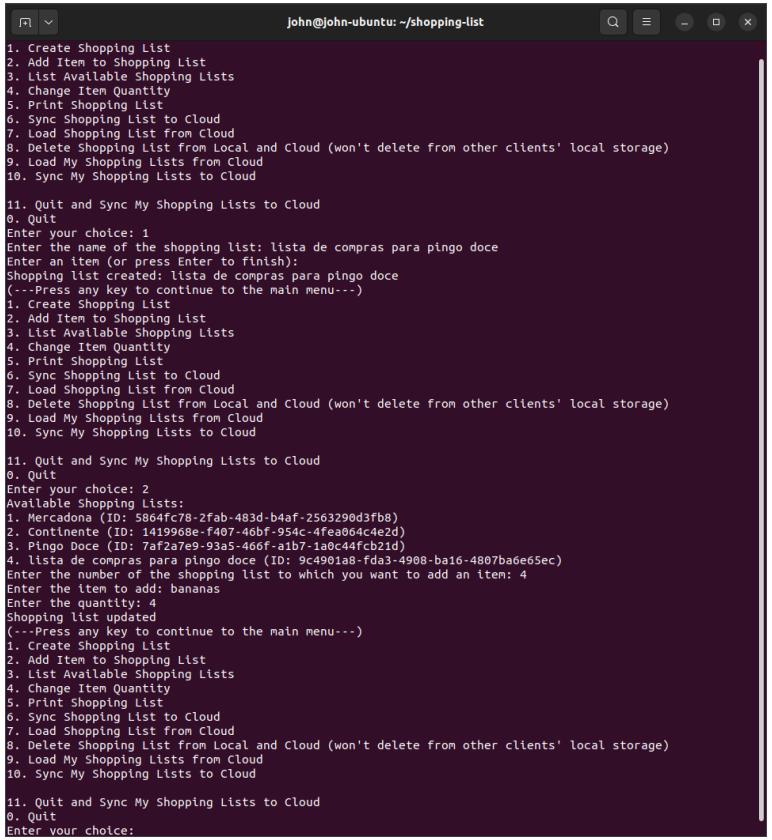
SS SHOPSYNC
The new shopping experience

- **Local-first**: The user can run the application without connecting to the server. The data persists in the client's machine after use
- **Cloud component**: There is a cloud component that allows users to share data among themselves and provide backup storage
- **Uniqueness**: A user can share the list, identified by a unique ID (uuid4). While the probability that a UUID will be duplicated is not zero, it is generally considered close enough to zero to be negligible.
- **List sharing**: A user can fetch and load shopping lists from the cloud on demand
- **Shopping list ownership**: If a user creates a shopping list or fetches a shopping list by its unique ID, the shopping list will appear on the user's shopping lists list as their own
- **List deletion**: A user can delete a shopping list from its storage and the cloud. Versions of that shopping list that are stored locally on other users will not be deleted, and they can still load the shopping list to the cloud if wanted
- **Version control**: The version control is made using CRDTs (Conflict-free Replicated Data Types), which allow merges of the shopping lists when two users alter the same shopping list

Two ways of using ShopSync: trough the web app or through the terminal

SHOPSYNC
The new shopping experience





Demo: main use cases on the web app

SS SHOPSYNC
The new shopping experience

SDLE 23/24

The new shopping experience

Demo: local first application – works with no connection

SS SHOPSYNC
The new shopping experience

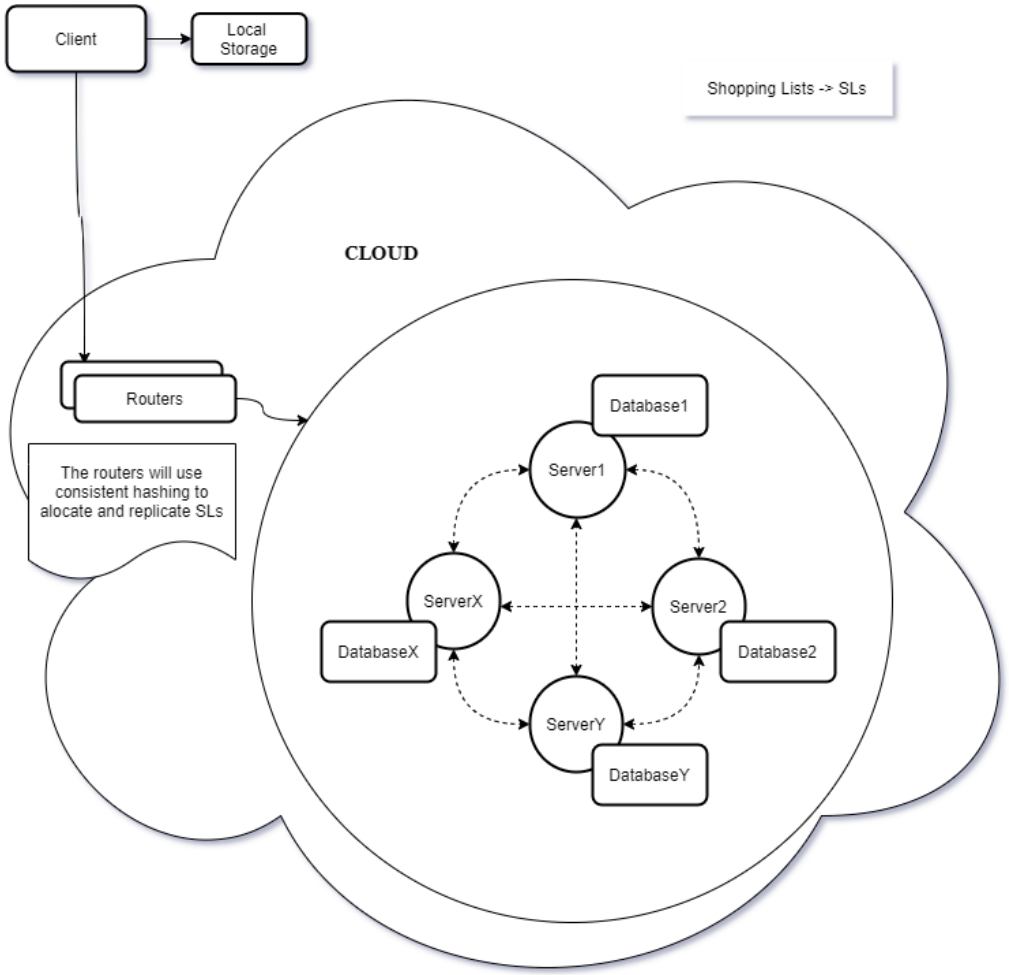
SDLE 23/24

The new shopping experience

The application is divided between three main components: Client, Router and Servers

SHOPSYNC
The new shopping experience

- Each client has a local storage unit
- Clients connect to a router
- The cloud side is divided into router and server.
- There are two routers, one primary and one other for backup
- The server is composed by a ring of nodes, where the nodes can all communicate with each other



The client-side application is composed of the user interface, the database, and the shopping list creation, editing, and deletion logic

SS SHOPSYNC
The new shopping experience

- **Interface**: As seen before, the client has available two interfaces: the first one being a web app, built using Flask and Angular (serves as the concept prototype for future implementation) and the terminal interface (serves as a more insightful experience for educational purposes)
- **Database**: The data is stored locally in a sqlite database. When the user starts the application, all the shopping list data is loaded, and when the client exits all the shopping lists are saved
- **Shopping list logic**: All the functions and objects needed for shopping list creation and manipulation
- **CRDTs**: The shopping list has an id and name and its items are stored as a map of CRDTs, with one entry for every item on the list. Our implementation of the CRDT is, in its essence, a PNCounter Map, adapted to suit the problem specifications

In our application, the client's username is defined by the port the client connects to. To connect to the cloud, the client must know the routers' addresses.

The router will use consistent hashing to choose which server node will coordinate the operation, as well as Binary Star Pattern to ensure availability

SS SHOPSYNC
The new shopping experience

- The router **serves as the middleman in the communication** between the client and the server. It is the router's main responsibility to exchange messages from the client to the respective nodes and vice-versa.
- **Start up**: After starting the router, it will check if any nodes are trying to connect to it and add them to its hash ring
- **Node monitoring**: Nodes are periodically monitored, to check on their health status, using a heartbeat. If a node is irresponsive, it is eliminated from the hash ring
- **High availability**: We ensure router high availability by implementing a Binary Star pattern for the routers, having two routers, one primary and one backup. Both routers, on start-up, will connect to the nodes in the server and will monitor them. However, only the active server will communicate with the client. The primary and backup routers exchange heartbeat status messages to share each other's status.
- **Consistent hashing**: The router passes the messages from the client to the server using consistent hashing. We used sha256 for this effect. The router send the message to the coordinator, and the coordinator is responsible to assign other nodes to participate in the quorum.

Source: ZMQGuide (2023)

Demo: Router crashing

SS SHOPSYNC
The new shopping experience

SDLE 23/24

The new shopping experience

The server consists of a ring of nodes, where every node can communicate with one another

SS SHOPSYNC
The new shopping experience

- **Start up:** on start, every node reads/creates its database, and then send a message to the routers to register itself on the network
- **Database**: every node of the server has one own sqlite database, to store information about shopping lists. The node updates the database on exit
- **Reliability**: to ensure that the data is as reliable as possible, we store the same information across various nodes. Decision are made by quorum, having multiple nodes participating in the voting's. When a node is asked for a list but does not have it, it asks another node to send it to them.
- **Scalability**: it is possible to, at any time, add nodes to the system, increasing the system's capacity if the number of requests increases
- **Node monitoring**: The nodes monitor each other, in other to know whether a node is alive or not, which is important when forming quorums, which are coordinated by a single node selected by the router.
- **Hinted Handoff**: When a node, which was supposed to receive a shopping list, is down at the time of a request, there is a node that saves that information and will try to send the information back to the node that is responsible for the data. If successful and the substitute node is not responsible for that list, it deletes it

Demo: Quorums/hinted handoff working

SS SHOPSYNC
The new shopping experience

SDLE 23/24

The new shopping experience

Demo: Quorums/hinted handoff working

SS SHOPSYNC
The new shopping experience

```
C:\Users\jcarv\Documents\FEUP\M1ano\SDLE\LargeDistributedShoppingList\venv\Scripts\python.exe
 C:\Users\jcarv\Documents\FEUP\M1ano\SDLE\LargeDistributedShoppingList\node.py 5564
coordinating quorums
listening for nodes
{'type': 'REGISTER_RESPONSE', 'nodes': ['tcp://localhost:5560', 'tcp://localhost:5561', 'tcp://localhost:5562'],
 'origin': <zmq.Socket(zmq.DEALER) at 0x1082a70b540>}
{'type': 'REGISTER_RESPONSE', 'nodes': ['tcp://localhost:5562', 'tcp://localhost:5560', 'tcp://localhost:5561'],
 'origin': <zmq.Socket(zmq.DEALER) at 0x1082a70b690>}
b'tcp://localhost:5560'
{'type': 'PUT_HANDED_OFF', 'key': '5117786e-54e4-4150-bbfa-4744b89fa069', 'value': '{"id":
 "5117786e-54e4-4150-bbfa-4744b89fa069", "name": "lista2", "items": {"counters": {"\\u00e7\\u00e7\\u00e7":
 {"inc_counter": {"counter_map": {"tcp://localhost:6002": 7}, "replica_clock": {"tcp://localhost:6002": 1}},
 "dec_counter": {"counter_map": {}, "replica_clock": {}}}}}}'}
```

Limitations

SS SHOPSYNC
The new shopping experience

As far as our work goes, there are still improvements to be made to ensure a more reliable, available and secure system. These are some of the limitations found:

- **Client needs to have the address for both routers to connect**: the principal router could be responsible to give the backup address, but what if the client never gets to connect with the principal router? Clients could also try and connect to one port from X to Y, and when they get a response assume that is the active router
- **CRDT could be a bounded counter:** using the bounded counter would be better to handle decrements for negative numbers, as we must check that using a PN counter
- **Synchronization from client to server is manual**: in the current state the client must indicate when he wants to sync the information to the server. It would improve the user experience if he could have this option
- **Client no overlapping actions**: the client is not able to make multiple request at the same time, it must wait for a response before proceeding to the following request
- **Router does not connect to nodes on reload**: when the server goes down, it cannot connect to the nodes. As of now, nodes only send register messages on startup. One possible solution to this would be for the node to send a register message periodically when the router is down, or the active router to send a message to the nodes when it starts receiving heartbeats from the other router again

SS
SHOPSYNC

The new shopping experience

THANK YOU!

Q&A