



Shopping Lists on Cloud

Large Scale Distributed Systems

Carlos Baquero-Moreno
Pedro Ferreira Souto

Gonçalo Pinto	up202204943
José Granja	up202205143
Leonardo Ribeiro	up202205144
Manuel Mo	up202205000



Index

1. Project Context and Overview
2. Technologies
3. User Interface
4. Local-first
5. CRDTs
6. Cloud
7. Main difficulties
8. References



Project Context and Overview

- **Local-First Approach:** Focuses on running code **directly on users' devices** to store data, allowing the app to function even **without an internet connection**. This improves the user experience by letting users **view** and **update** their **shopping lists offline**.
- **Shared Shopping Lists:** Every **shopping list** is assigned a **unique ID**, making **collaboration** easy. Users who have the **list ID** can **work together**, managing and updating items **in real-time**.



Technologies

The application was developed with an emphasis on **clarity**, **independence** and **accessibility**, leveraging **lightweight** and **reliable technologies** to simplify ongoing usage. The core components include:

- **Frontend:** React (TypeScript), Tailwind CSS
- **Backend:** Node.js, Express.js
- **Database:** SQLite (server), IndexedDB (client)

To enable **scalable communication** between system components and to preserve **consistency** in **distributed environments**, the following technologies were incorporated:

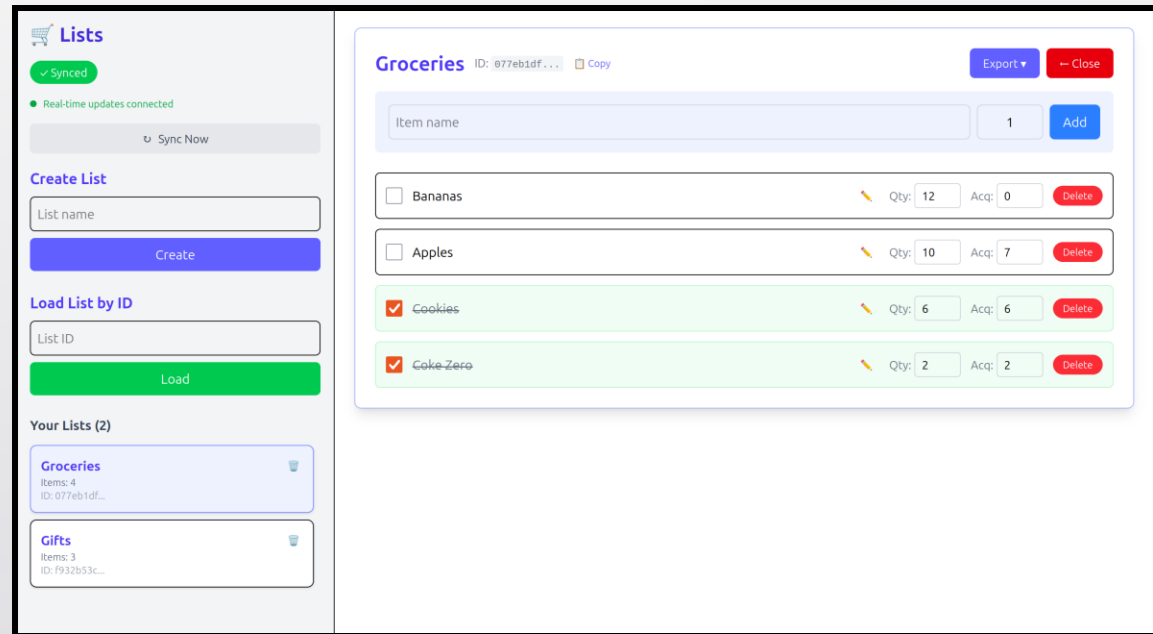
- **Sync & Communication:** REST API, Server-Sent Events (SSE), UUID to ensure globally unique identification across the system and ZeroMQ for server-to-server gossip and for efficient asynchronous message exchange
- **CRDTs:** Custom TypeScript implementations (PNCounter, LWWRegister, AWORSet, VectorClock)

User Interface

- **Modern, intuitive** and **responsive UI**, built with React + Tailwind CSS, and capable of executing the **CRUD operations** required for the project:

Features:

- **Shopping Lists:** create, delete, load and export;
- **Items:** add, delete, rename and check/uncheck;
- Real-time updates via **SSE**;
- **Offline Support** and **Automatic Sync**



Local-First

The client application is designed with a **local-first approach**:

- All changes are immediately applied to **local IndexedDB** for **instant feedback** and **offline support**.
- Changes are **queued** and **synchronized** with the **server** when **connectivity is available**.
- **SyncService** manages **conflict resolution** and **merging** using **CRDT logic**, ensuring a smooth user experience regardless of network state.

The screenshot displays a web application interface. On the left is a sidebar titled 'Lists' with a shopping cart icon. It shows a 'Queue' of 4 items, a 'Sync Now' button, and a 'Create List' section with a 'List name' input and a 'Create' button. Below that is a 'Load List by ID' section with a 'List ID' input and a 'Load' button. At the bottom of the sidebar, under 'Your Lists (2)', a 'Groceries' list is shown with 5 items and ID: 077eb1df... A yellow banner at the bottom of the sidebar states 'You're offline. Your changes are saved locally and will sync when you reconnect.'

The main area shows a 'Groceries' modal window. It has a title 'Groceries' with an ID '077eb1df...' and a 'Copy' button. There are 'Export' and 'Close' buttons in the top right. Below the title is an input field for 'Item name' and a quantity input '1' with an 'Add' button. The list contains four items: 'Flour' (unchecked), 'Bananas' (checked), 'Apples' (checked), and 'Cookies' (checked). Each item has a quantity input, an 'Acq' input, and a 'Delete' button. The 'Bananas' item has a quantity of 12 and an 'Acq' of 12. The 'Apples' item has a quantity of 10 and an 'Acq' of 10. The 'Cookies' item has a quantity of 6 and an 'Acq' of 6. The 'Coke Zero' item has a quantity of 2 and an 'Acq' of 2.



CRDTs

We implemented CRDTs (Conflict-free Replicated Data Types) to maintain consistency and correctness in a distributed environment without requiring centralized coordination, ensuring safe merging of concurrent changes. The main CRDT implementations were:

- **AWOR-Set (Add-Wins Observed-Remove Set):** to handle item addition and removal, ensuring that concurrent additions are preserved even in the presence of removals;
- **PN-Counter:** to manage item quantities, allowing both increments and decrements while maintaining convergence across replicas;
- **LWW-Set (Last-Writer-Wins Set):** to store and resolve updates to item names, ensuring that the most recent update is consistently applied.



Cloud

Implemented using the **ZeroMQ** library, the system utilizes **REQ/REP sockets** along with the **Lazy Pirate** pattern to create a robust **request-reply** messaging layer.

Communication follows this flow:

- **Client** → **Storage Node (HTTP API)** – CRUD operations on lists/items.
- **Storage Node** → **Neighbors (ZeroMQ)** – Gossip protocol for replication & hinted handoff.
- **Storage Node** → **Coordinator (ZeroMQ)** – Forward updates for real-time broadcasting.
- **Coordinator** → **Clients (SSE)** – Push live updates to all connected frontends.



Cloud

- **Dynamic Connection Handling:** Connections automatically adjust between clients and servers based on availability through **consistent hashing** failover and **quorum-based read/write** adaptation, providing greater flexibility and efficiency.
- **Automated Server Recovery:** Servers recover independently after failures through **deterministic ring rejoining (consistent hashing)** and **hinted handoff** delivery coordinated with **quorum operations**, improving overall system reliability.
- **Seamless Client Interaction:** Users work effortlessly without needing cloud infrastructure knowledge, thanks to **consistent hashing** that automatically routes requests to the correct **replicas** and **quorum operations** that ensure **data availability**.



Main difficulties

- Consistency across clients when there are many concurrent changes;
- The late implementation of the quorum created problems with the lists each customer had access to, and with the synchronization between clients;
- Item check consistency between the check via checkbox and check via quantity and acquired update (merge difficulties).



References

- [IndexedDB](#)
- [React](#)
- [ExpressJS](#)
- [UUID](#)
- [Local-First](#)
- [CRDTs](#)
- [ZeroMQ](#)
- [Dynamo](#)