# SDLE

Group QuantumList

November 2023

## 1 Introduction

In a world where 7 billion people live, data ownership, privacy, and user control are paramount concerns in our society. This project is designed with these ideals in mind so that, the architected solution adheres to the principles of "local-first". This approach not only empowers users with full ownership of their data but also offers the flexibility to share and edit it on their terms. In this document, we will explore the core elements of our architecture aligning with seven key ideals: fast, multi-device, offline access, collaboration, data longevity, privacy, and user control. All this while maintaining high availability so data is accessible seamlessly from around the world without user concerns.

## 2 Architecture

### 2.1 Local-First

The project's architecture will follow the local-first approach, giving full ownership of the data to the user, while allowing the data to be shared and edited. The user can create shopping lists existing only on his machine and editable only by him, whenever he decides to, the shopping list will be shared, presenting him an ID that can be used to share and access. A user-created shopping list will only be updated whenever the user requests it, being notified if there are new changes to the online version or not. To prevent the user from losing their local copy whenever he chooses to synchronize, he can create his non-synchronized copy of the shopping list.

We aim to fulfill as much as possible the seven ideals of local-first: fast, multi-device, offline, collaboration, longevity, privacy, and user control.

### 2.2 Scalability

To achieve scalability, our architecture employs a straightforward strategy. For increased scalability within a given region, we can add more sub-regions as needed. These sub-regions are equipped with load balancers to evenly distribute the workload.

In addition, our use of Leaderless Replication allows us to easily scale the system by adding more nodes within each sub-region. Furthermore, we can enhance scalability by incorporating additional data centers into the network, facilitating the distribution and availability of data across multiple geographical locations. This ensures that the system can efficiently grow in response to increased demand and user needs.

### 2.3 Replication and Availability

The architecture will be using the Leaderless Replication approach with Sloppy Quorum Consensus, ensuring data is stored in multiple nodes, in order to have high availability. The need for multiple nodes is due to the possibility of one falling, having its data replicated on another node prevents losing the data or making it unavailable to the user. Leaderless Replication provides multiple nodes without the bottleneck of having the data be written on one single node first and providing less latency however, one node failing or being slow, adds latency to the system.

In order to increase availability we have distributed load balancers so that it is almost guaranteed a server ready to process the requests. And we also use CDNs.

Using the Leaderless Replication approach on a concurrent system leads to a consistency problem that requires a consistency mechanism.

### 2.4 Consistency

To deal with the consistency problem, CRDTs will be used to improve consistency by enabling multiple replicas to converge to the same state.

Delta CRDTs with vector clocks will be implemented both on the local app side and on the database side. Locally they deal with the list operations, on the database side it is where the operations of all clients will be merged. Vector clocks allow you to deal with concurrent operations such as add and delete or two adds. For delete, tombstones are used in order to avoid any add operation while the database and clients do not have the same state, where the tombstone is then deleted. Two add, add the sum of the quantities.

Using Quorum also comes with Replication Lag, we solve this by having the orchestrator ensure Monotonic Reads.

### 2.5 Data Partitioning and Distribution

Data will be partitioned inside a cluster using key range partitioning, the key ranges for each node will be dynamically re-balanced when a partition gets too big or when new nodes are added to the cluster, the keys are defined using the shopping list id.

### 2.6 Fault Tolerance

The chosen architecture takes fault tolerance into consideration having active and passive components on potential single points of failure so that the passive can take over the active's spot in case of failure.

## 3 Implementation

The application interacted by the user will be built using Tauri [6], a toolkit that enables the creation of desktop applications using virtually any framework used on a browser, such as SvelteKit [4] and TailwindCSS [5], while also allowing the use of a backend is Rust which is faster than using web workers. Tauri was chosen because we do not want the application to rely on an internet connection and a desktop application can be used locally without the need of a local web server and not depending on the browser.

The server-side components will be developed using GoCZMQ [2], which is a Golang wrapper for CZMQ [1], a high-level C binding for ZeroMQ [7].

The data storage, both locally and on the server, will be done using SQLite [3] because it is a simple and lightweight database that does not require any additional setup.

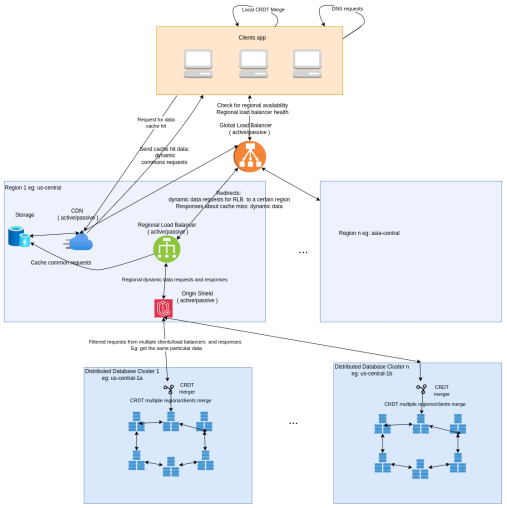


Figure 1: System Architecture

## 4 Conclusion

We aim to fulfill as much as possible the seven ideals of local-first: fast, multi-device, offline, collaboration, longevity, privacy, and user control. With this in mind, we believe our architecture balances all criteria evenly while maintaining consistency and simplicity.

## References

[1] Czmq. http://czmq.zeromq.org/.

[2] Goczmq. https://github.com/zeromq/goczmq.

[3] Sqlite. https://www.sqlite.org/index.html.

[4] Sveltekit. https://kit.svelte.dev/.

[5] Tailwindcss. https://tailwindcss.com/.

[6] Tauri. https://tauri.app/.

[7] Zeromq. https://zeromq.org/.