Local First Shopping Lists

Large Scale Distributed Systems

Carlos Veríssimo - up201907716
João Felix - up202008867
José Costa - up202004823

Introduction

Goal: Develop a scalable, real-time, cross-platform shopping list application

Key Features:

- Offline-first design
- Real-time synchronization
- Conflict resolution for concurrent edits
- Sharing your list with other users

Technologies: React, NestJS, TypeScript, Prisma, PostgreSQL, ZeroMQ, CRDTs

Frontend Architecture

Technology Choices:

- **React:** Modular and reusable UI components.
- **IndexedDB** with Dexie.js **:** Offline-first design.
- **Fetch API / Axios:** HTTP requests for backend communication.
- **Socket.IO:** Real-time updates.
- **JWT Authentication:** Secure API access.

Key Components:

- Authentication Form
- Shopping List UI (+ viewing other user’s lists)
- Sync Service

Challenges:

- Managing offline and online synchronization.
- Ensuring real-time responsiveness with ZeroMQ.

“Proxy” Architecture

Why?

ZeroMQ is not directly compatible with web browsers, while Socket.IO is specifically designed for web applications.

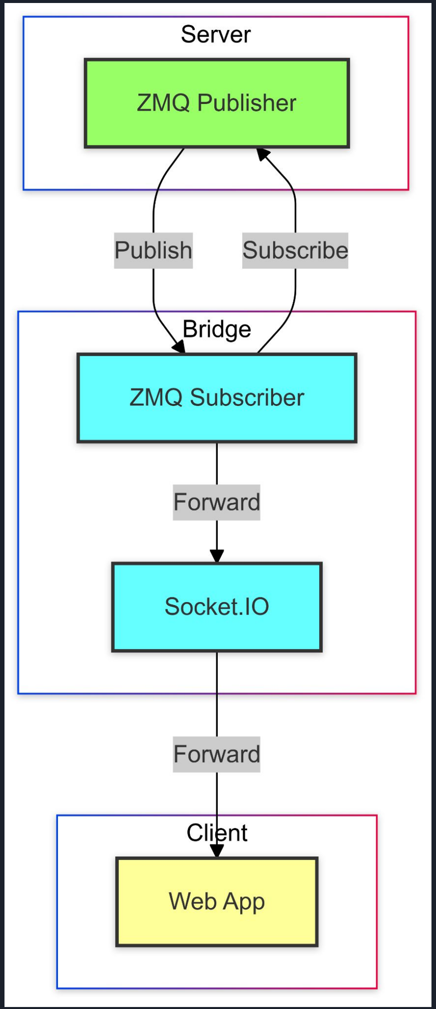
Capabilities

The bridge allows the backend to use ZeroMQ's powerful pub/sub capabilities while still maintaining real-time communication with web clients.

What it does

The bridge ensures messages are properly routed from ZeroMQ publishers to the correct Socket.IO clients by:

- Parsing messages from ZeroMQ
- Forwarding them to specific user rooms in Socket.IO
- Handling error cases



Backend Architecture

Technology Choices:

- **NestJS:** Structured, modular API backend.
- **Prisma ORM:** Simplified database interaction with PostgreSQL.
- **ZeroMQ PUB/SUB:** Real-time synchronization.
- **CRDTs:** Handling concurrent updates.
- **JWT Authentication:** Secure, stateless user sessions.
- **Redis and Bull:** for queuing and processing jobs

Key Modules:

- **Auth Module:** Registration, login, password hashing.
- **ShoppingList Module:** CRUD operations and conflict resolution.
- **Sync Module:** Real-time updates via ZeroMQ.

Challenges:

- Efficient conflict resolution for concurrent updates.
- Scaling ZeroMQ for multiple clients

Database Design

Database: PostgreSQL (Supabase)

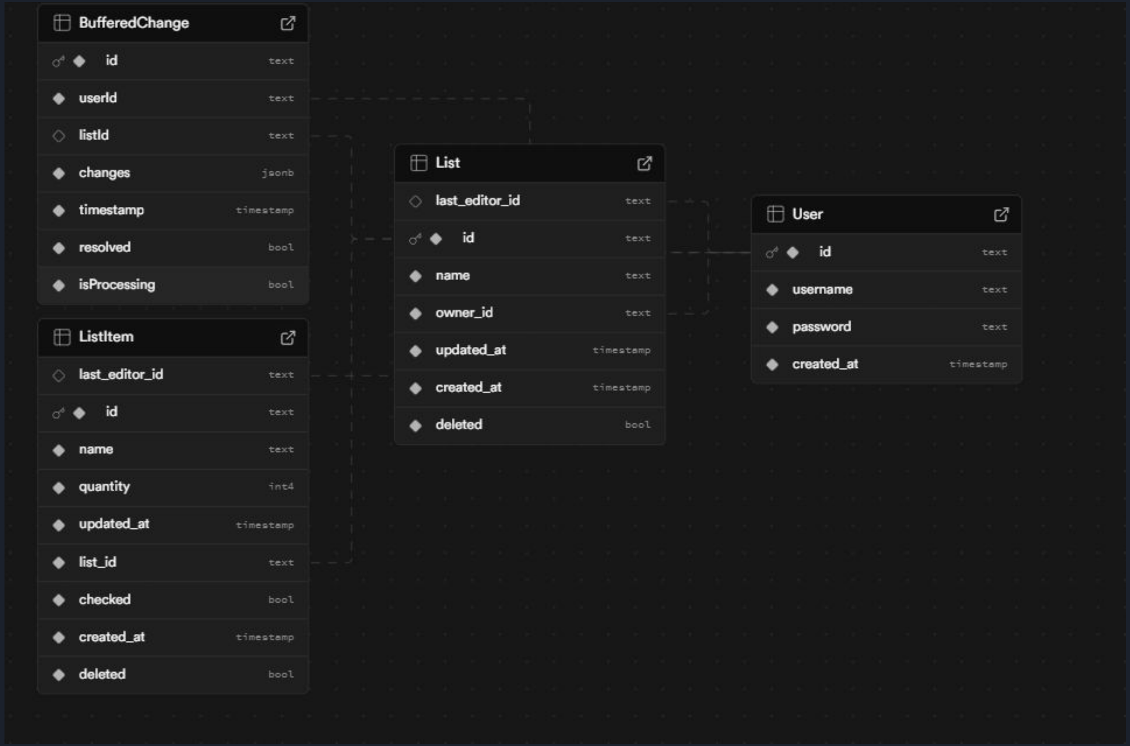
Structure:

- **Users Table:** User credentials and metadata.
- **ShoppingLists Table:** List metadata (ID, name, owner, last editor).
- **ListItems Table:** CRT-compatible identifiers, quantity, and status.
- **BufferedChanges Table:** Stores list changes to be later processed by the CRDT

Challenges:

- Ensuring consistency across distributed replicas.
- Optimizing performance for real-time updates.

Database Design



Conflict Resolution and Concurrency Handling

Conflict Resolution Strategy: Last-Writer-Wins (LWW)

- Changes are aggregated by list. The latest change to an item of the list, as well as other changes such as renaming or deleting the list, is the one to be replicated across all replicas.

Concurrency Handling:

- Offline modifications on replicas (devices).
- Changes are buffered until processed by the CRDT consumer.
- Backend reconciliation for (almost) real-time consistency.

Challenges:

- Designing robust CRDTs for complex scenarios.
- Minimizing latency for conflict resolution.

Data Sharding and Replication

Database Sharding

- Distributes Data across multiple servers;
- Each server is represented by 4 virtual nodes (20 total) to ensure better distribution;
- Enables horizontal scaling and handling of large volumes of data..

Consistent Hashing:

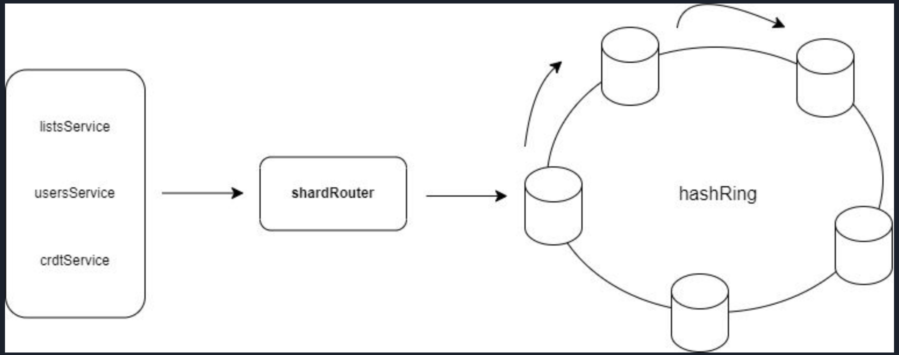
- **D**istributes data evenly according to **userId hash.**

Replication:

- Creates multiple copies of data to **N(3)** shards.
- Provides **redundancy** in case of shard failure.

Read/Write Quorum:

- Requires a majority of replicas to agree on read/write operations (**W=R=2**)
- Ensures Data **consistency** and **integrity** in the case of partial failures.
- Overlap condition is guaranteed (**W + R > N**).



Design Challenges

Frontend:

- Ensuring smooth offline-to-online transitions.
- Optimizing ZeroMQ real-time updates.

Backend:

- Implementing CRDTs for conflict-free concurrent updates.
- Scaling ZeroMQ for a growing number of users.

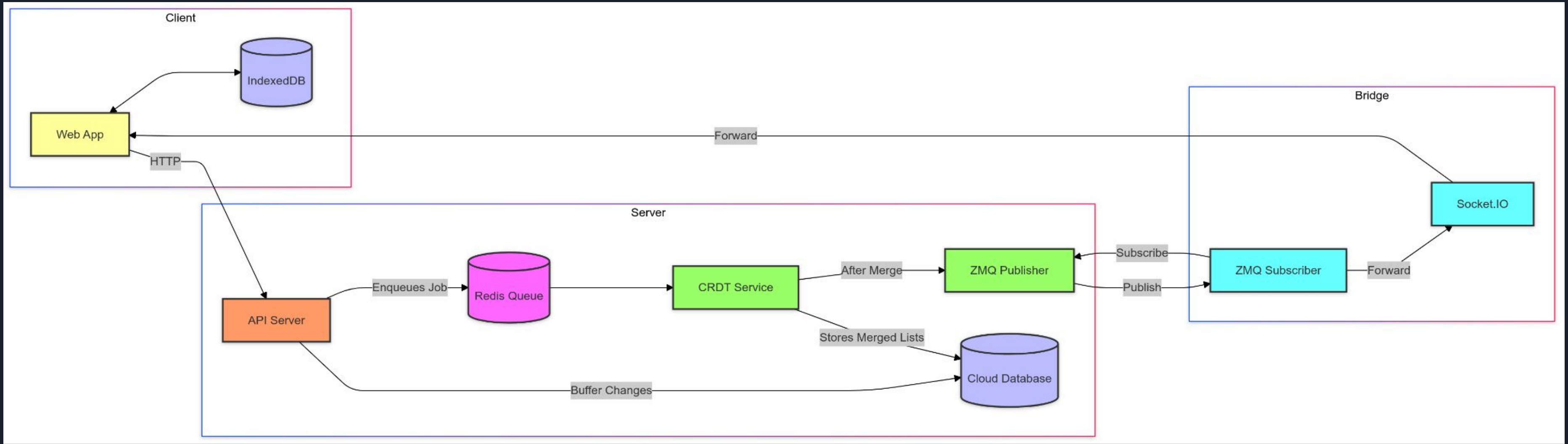
Database:

- Efficient sharding for large-scale data.
- Maintaining consistency across distributed replicas.

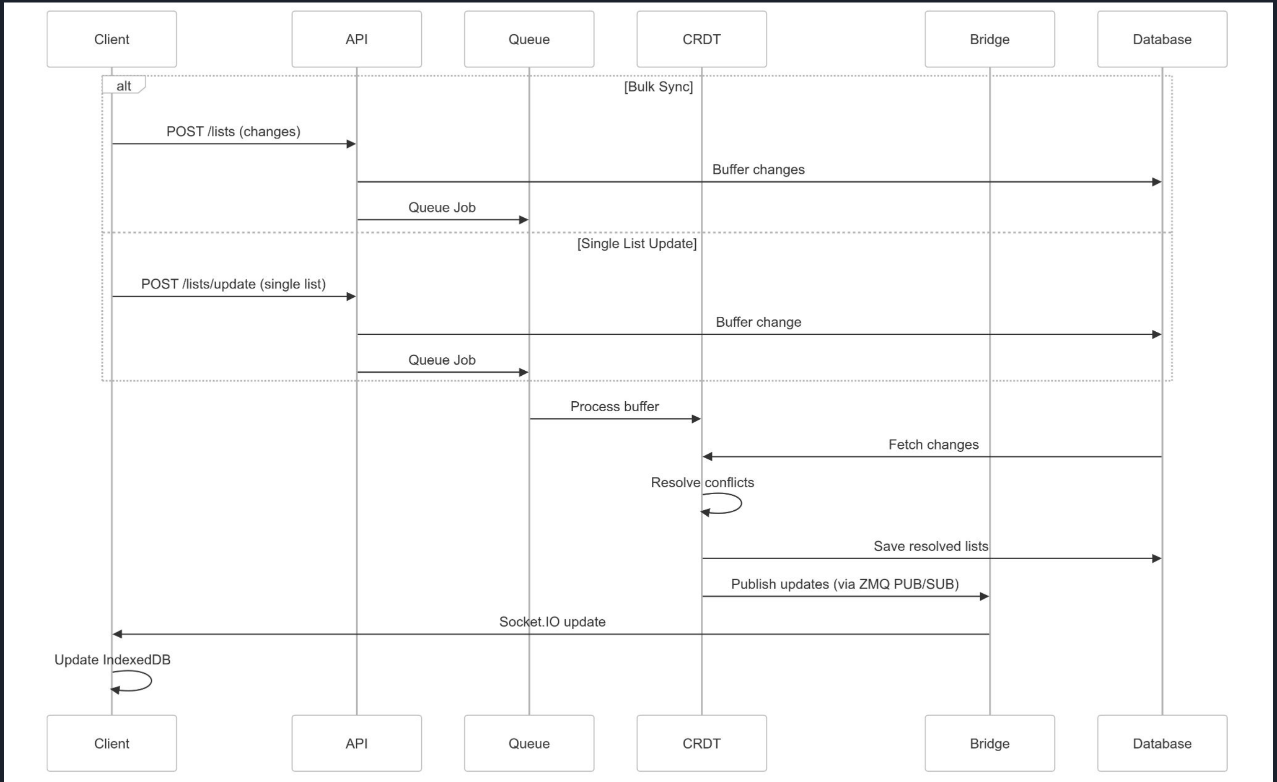
Overall:

- Balancing simplicity (LWW) and robustness (CRDTs).
- Ensuring low-latency synchronization.

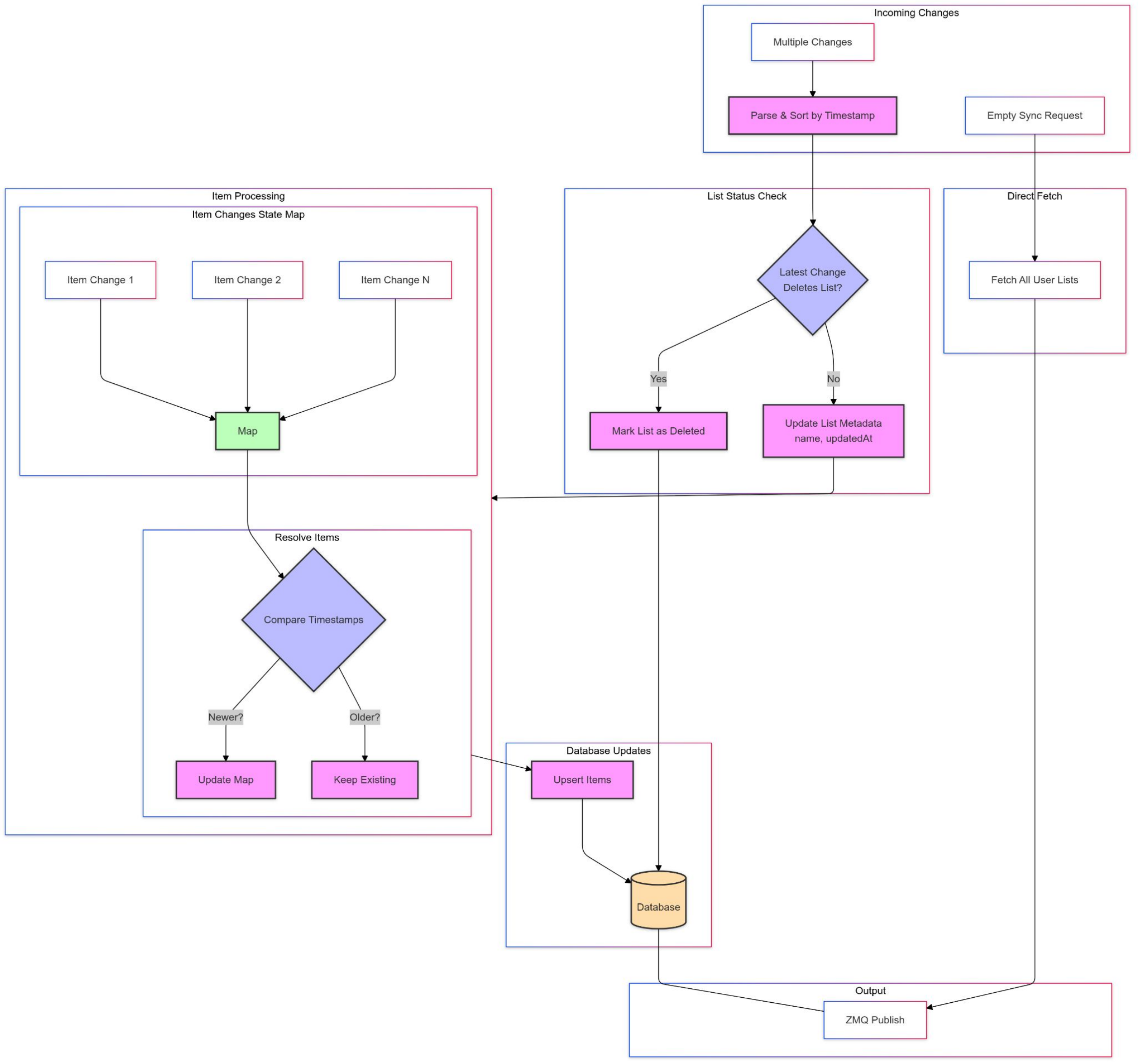
System Diagram



Sync Flow Diagram

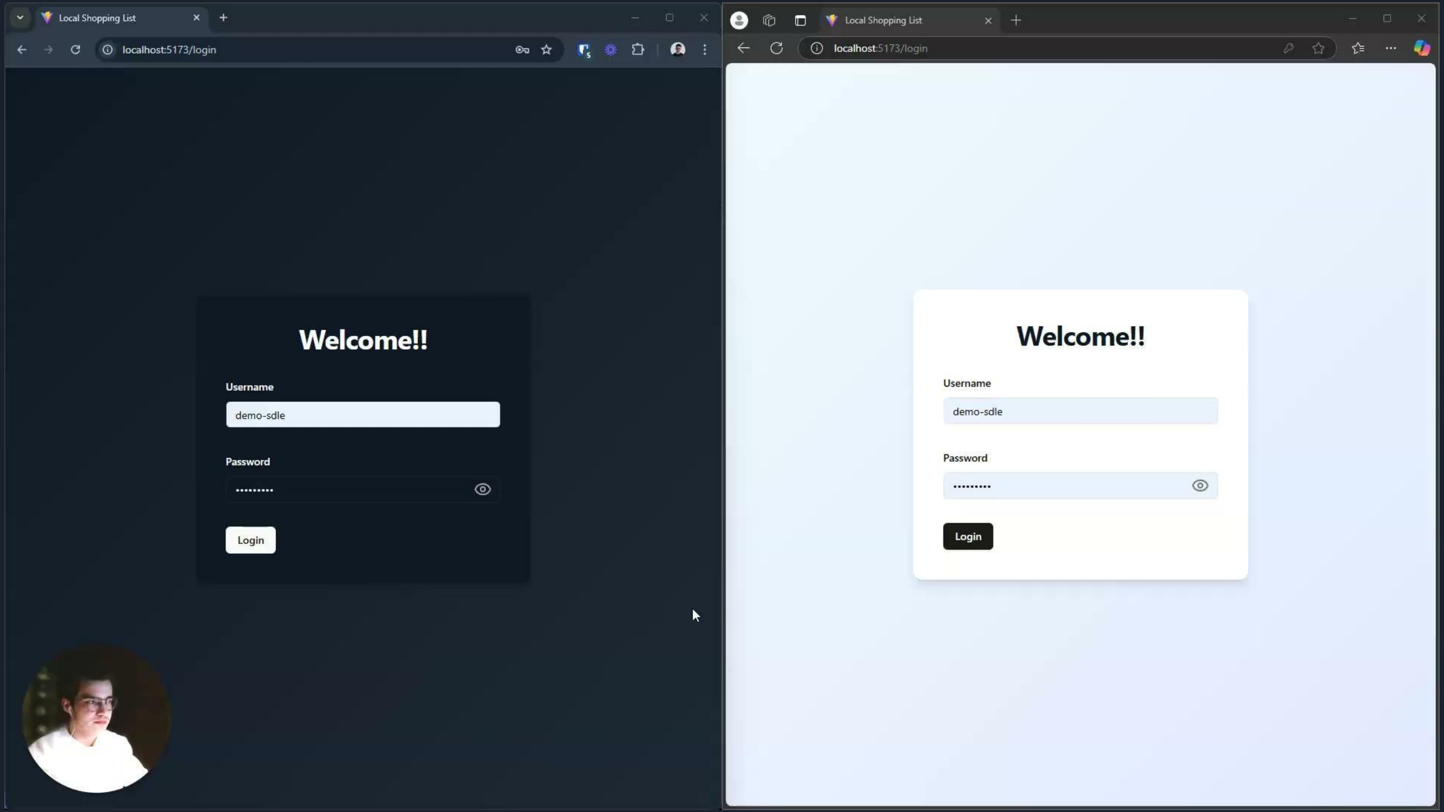


CRDT Flow



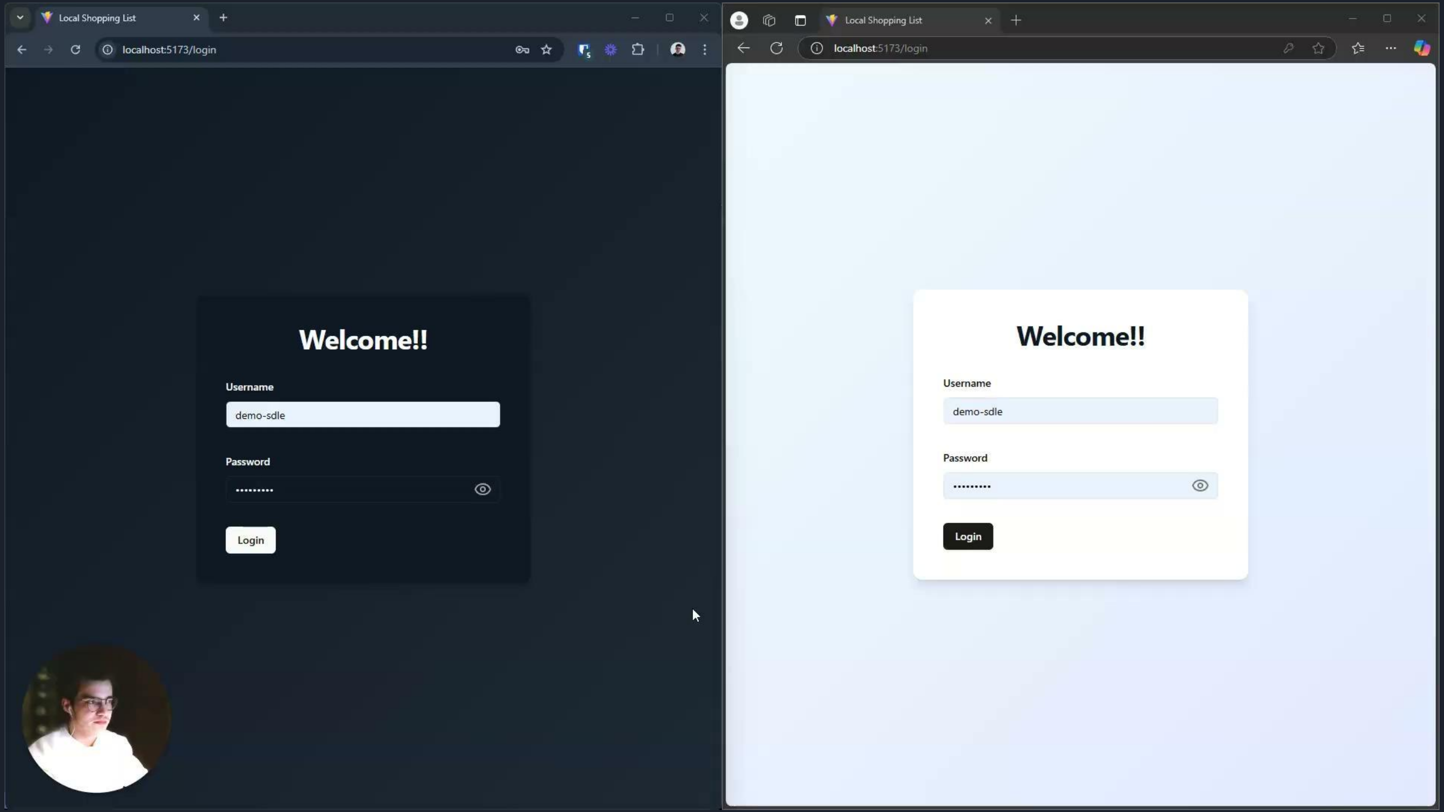
Incoming Changes
Multiple Changes
Parse & Sort by Timestamp
Empty Sync Request
Item Processing
Item Changes State Map
Item Change 1
Item Change 2
Item Change N
Map
List Status Check
Latest Change Deletes List?
Yes
No
Mark List as Deleted
Update List Metadata name, updatedAt
Direct Fetch
Fetch All User Lists
Resolve Items
Compare Timestamps
Newer?
Older?
Update Map
Keep Existing
Database Updates
Upsert Items
Database
Output
ZMQ Publish

Use Case 1 - Offline Capability



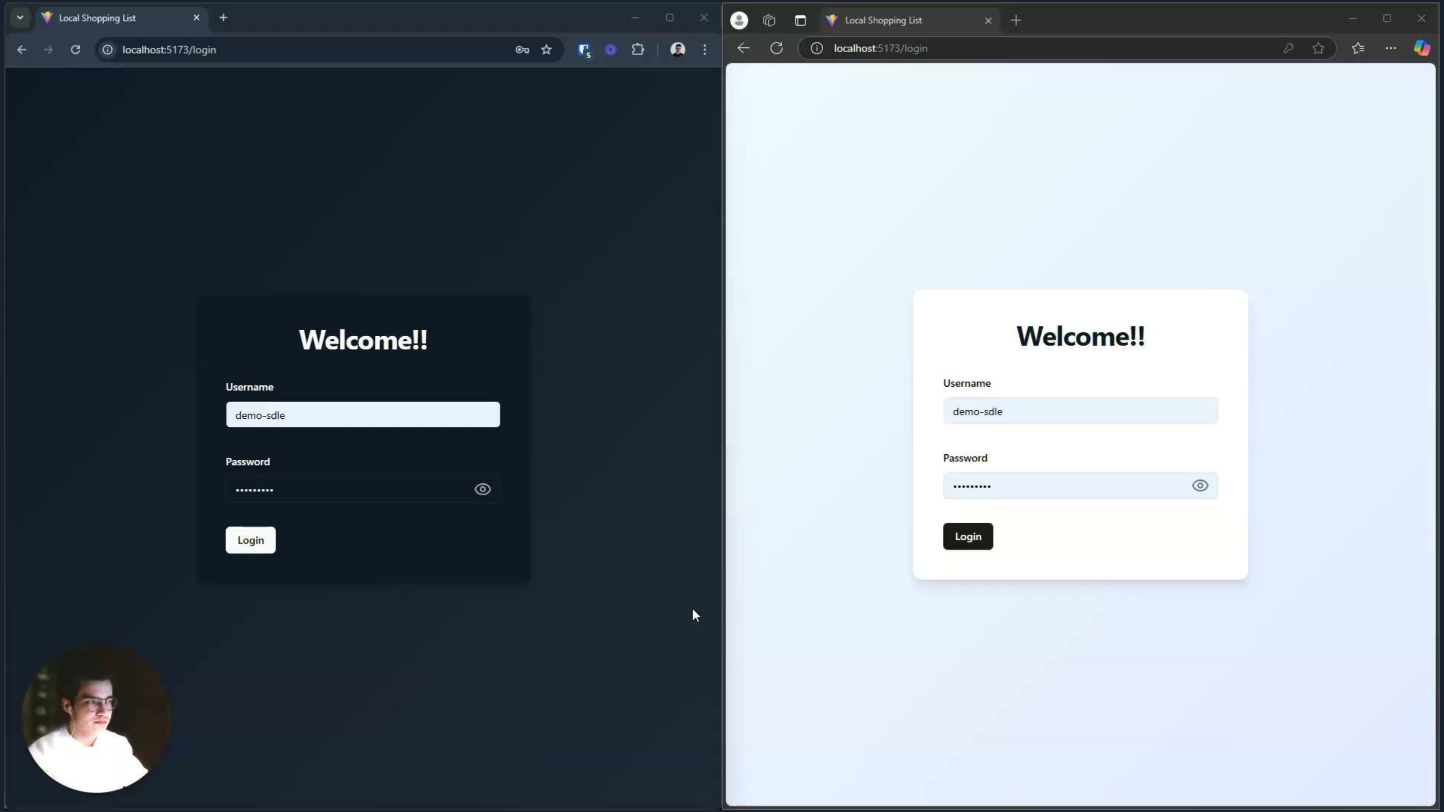
View in loom

Use Case 2 - Concurrent Updates



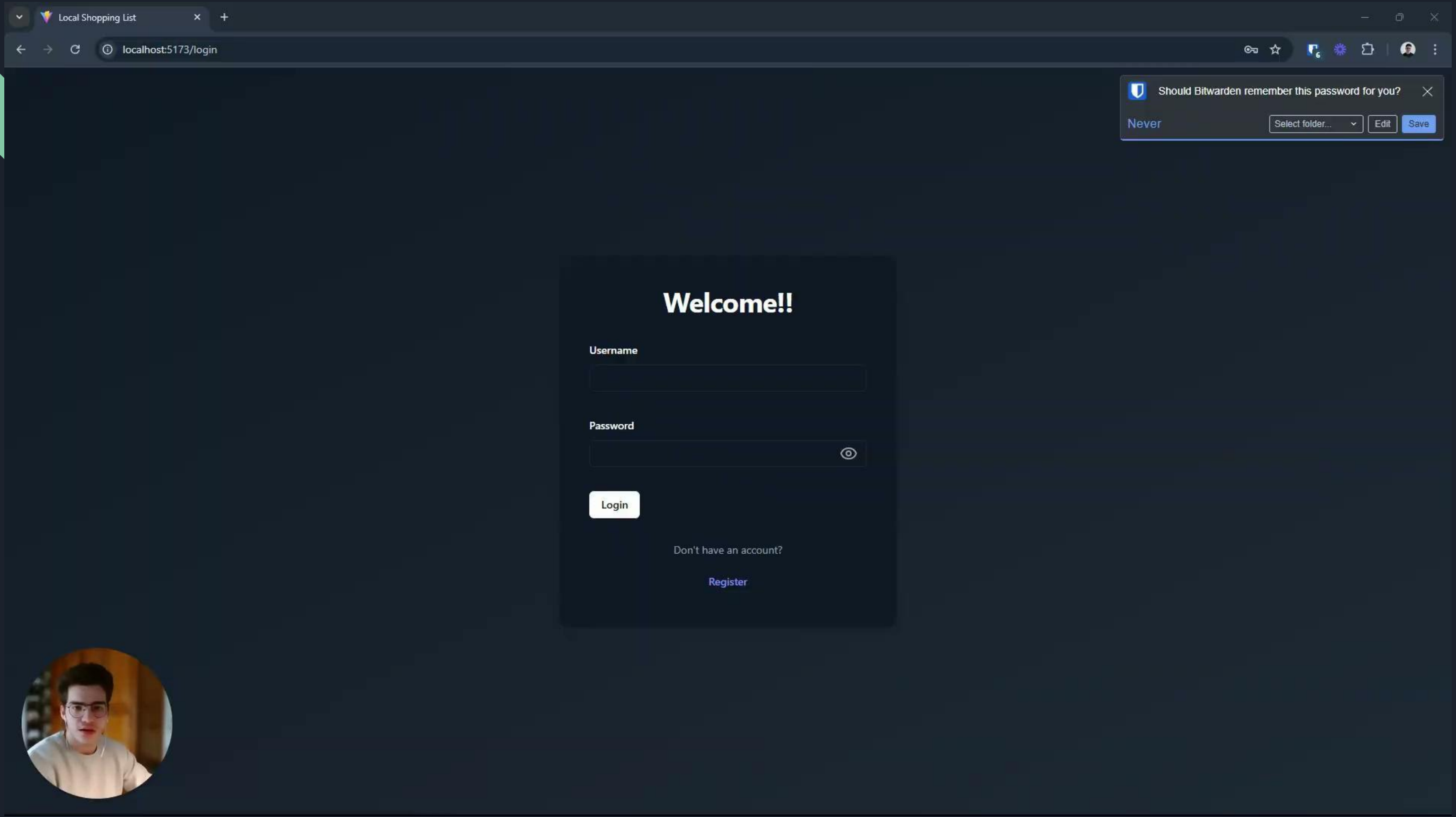
View in loom

Use Case 3 - Changes by Other Users



View in loom

Demo



View in loom