U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Shopping Lists on the Cloud

Large Scale Distributed Systems

Afonso Martins - up202005900
Anete Pereira - up202008856
Eduardo Silva - up202005283
Hugo Castro - up202006770

INTRODUCTION

- Explore the development of a **local-first shopping list application**.
- Code runs on user devices for local data persistence.
- Cloud component facilitates data sharing and backup storage.
- Users create and manage shopping lists via a user-friendly interface.
- Use Conflict-free Replicated Data Types (CRDTs) for robust consistency.

REQUIREMENTS

- **Local-First Approach** - The application should be designed with a local-first approach, allowing users to persist data locally on their devices.
- **Cloud Component** - Include a cloud component that enables users to share shopping lists with others and provides backup storage.
- **Unique List ID** - Each shopping list should have a unique identifier that can be shared with other users
- **User Permissions** - Users with the list ID should be allowed to add and delete items on the list
- **User Interface** - Provide a user interface for users to create, modify, and delete shopping lists
- **Concurrency Support** - Support concurrent changes to shopping lists by multiple users
- **Replication and Consisty Management** - Application should explicitly handle the replication of the shopping lists, including the consistency of the replicas.

TECHNOLOGY

MERN

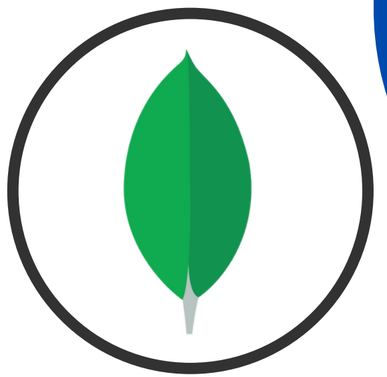
Chose MERN stack (MongoDB, Express.js, React, Node.js) for familiarity and rapid development.

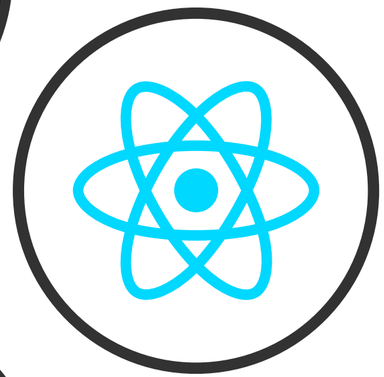
TYPESCRIPT

Provides strong typing, improves code quality, and boosts developer productivity in MERN stack development through better tooling and error prevention.

POUCHDB

Enables offline data storage and synchronization on the client side.

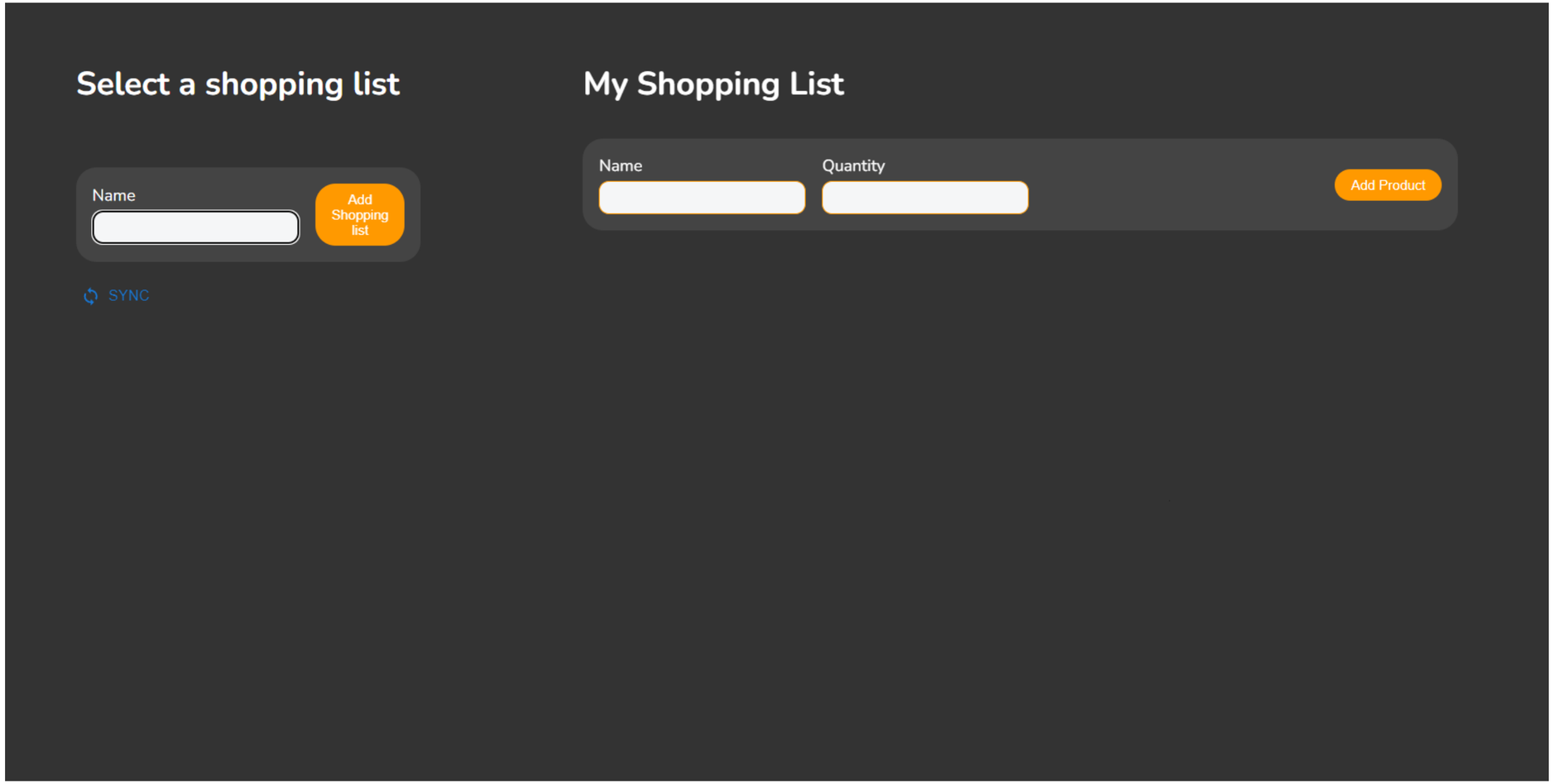




CLIENT

- All shopping list interactions occur on the **client side**.
- CRDT for local data handling enabling **concurrent changes and maintaining consistency**.
- Users can interact with the shopping list **offline**, and changes are stored in the local CRDT.
- **Local CRDT is stored in local database PouchDB.**

USER INTERFACE (1/2)



USER INTERFACE (2/2)

ADD SHOPPING LIST

Users can easily create new shopping lists via the dedicated button in the interface.

ADD PRODUCT TO A SHOPPING LIST

Convenient button for users to add products to a shopping lists previously chosen in the interface.

SYNC

Initiates a synchronization request to the server for seamless data transfer.

CRDT IMPLEMENTATION (1/2)

- **The CRDT implementation is based on professor Carlos Baquero delta-enabled CRDT's**
- **DOTS**

  Represent changes in data replicas, encompassing actions like incrementing or decrementing.
- **DOTKERNEL BASE**

  Includes essential functions: join, dotAdd, rmvDot, and more. Join for ORMap is custom, marking a more complex implementation.
- **DOTCONTEXT FOR VERSIONING**

  Implemented **DotContext** within **DotKernel**, acting as a historical record of Dots enables version comparison of CRDTs for effective join operations.

CRDT IMPLEMENTATION (2/2)

CCOUNTER

Casual Counter

Implements join operations to handle concurrent changes, ensuring conflict-free updates.

Supports **incrementing and decrementing operations** without conflicts.

Used to **track the quantity of items** in a shopping list.

ORMAP

Observed Remove Map

Implements customized join operations to manage concurrent changes in key-value pairs.

Supports the **addition, modification, and removal of entries**.

Used to **represent a shopping list** (dynamic list of items), with CCounters as values

SYNCHRONIZATION PROCESS

- Users trigger a **synchronization request** to the server by clicking the button available in the user interface
- Server responds by **sending** the Conflict-free Replicated Data Type (**CRDT**) stored in the cloud database as a JSON object
- **Client** performs a **join operation** between the local and cloud CRDTs.
- **Updated CRDT** version is **stored** both in the **local** and **cloud** databases, ensuring consistency across devices and databases
- JSON objects are serialized and de-serialized using a custom function

SERVER SIDE ARCHITECTURE

LOAD BALANCER

To efficiently distribute server accesses and ensure optimal resource utilization, the application incorporates a load balancer.

Two servers connected by proxy

Load balancer employs a **round-robin algorithm** for distributing connections among servers. This strategy involves each server handling incoming requests in a consecutive manner.

This even distribution keeps any single server from becoming a bottleneck, which improves system speed, resource utilization, and overall reliability.

SOLUTION EVALUATION (1/2)

EASY TO USE INTERFACE:

Simplifies interactions, making the application more accessible and enjoyable, contributing to an overall positive experience for users.

ABILITY TO DEAL WITH CONCURRENT UPDATES WITH CRDTS.

Efficiently managing simultaneous updates with CRDTs ensures data consistency and a reliable user experience.

SOLUTION EVALUATION (2/2)

LACK OF LIST SHARDING:

Lack of list sharding concentrates all updates through a single point, **potentially causing scalability issues**

Absence of sharding may lead to **increased response time**

A failure may affect another shopping list

LACK OF SERVERS AUTO-GROW MECHANISM:

Without auto-scaling, the system may **struggle to adapt to varying loads.**

In periods of **high demand**, the lack of additional servers could result in **decreased performance and responsiveness.**

May lead to **inefficient resource utilization** during **low-demand periods**.

CONCLUSIONS

- Exploring the development of a **local-first shopping list application** allowed us to gain a better understanding of other related themes.
- Explored the implementation of **Conflict-free Replicated Data Types (CRDTs)** for managing concurrent updates, a new data type previously unknown to us
- Developed strategies for **conflict resolution in concurrent updates**, acknowledging the importance of maintaining data consistency.
- Explored the development of **custom serializers and de-serializers**
- Learnt about **techniques of load balancing** and **server architectures** focused on **consistency and availability**