


# Shopping Lists on the Cloud

By:

- Alexandre Correia (up202007042@fe.up.pt)
  - Eduardo Duarte (up202004999@fe.up.pt)
  - João Félix (up202008867@fe.up.pt)
- 

# Project Overview

The main goal of our project is to develop a shopping list application that integrates both local and cloud components. This approach allows users to enjoy the advantages of local data persistence on their devices while also harnessing the collaborative power of the cloud.

## 1. Local Component:

- User Empowerment: Our application prioritizes a local-first philosophy, giving users control over their data.
- Offline Functionality: User's can create, modify, and access their shopping lists even without an internet connection.
- Data Autonomy: The local component ensures that user's shopping lists are persistently stored on their devices, offering autonomy and access.

## 2. Cloud Component:

- Seamless Collaboration: The cloud component facilitates collaboration by allowing users to share their shopping lists with others.
- Backup and Security: Users benefit from a cloud-based backup storage, enhancing data security and preventing loss.
- Universal Access: Cloud collaboration enables users to access and manage their shopping lists from multiple devices, ensuring a seamless and unified experience.

# User Interface

## Your Shopping Lists

Grocery List

Recipe List

Joao List

ta

Go Online



## Homepage

Here the user can access, and create their shopping lists

## Recipe List

### Needed

- **Tomatoes** - Needed: 4 ☐ Bought - 3 +

### Done

- **Chicken** - Needed: 1 ☒ Bought



## Shopping List

Within the list page the user can change the status of items and even add new ones

## Recipes

Recipe List



## Recipes

This is where the user can see the recipes made by the developers and create copies as their own shopping list

# Data Model

## List Properties

In our shopping list application, each created shopping list is uniquely identified by a system-generated ID. This ID serves as a distinctive reference point for the list, akin to a URL, making it easy for users to share and collaborate on specific lists.

## Item Properties

Bought Quantity:

- Purpose: Items in the shopping list have a bought quantity, specifying the current purchased quantity.
- Functionality: The bought quantity helps users understand how many items are still needed on that list. As items are acquired, the bought quantity increases in real-time for other users.

Quantity:

- Purpose: Items in the shopping list can be associated with a target quantity, specifying the intended quantity to purchase.
- Functionality: The target quantity feature assists users in planning their shopping by setting goals for the quantity of each item.

# Proxy

Servers can be added dynamically and lists are distributed among servers using consistent hashing. The proxy's ability to redirect clients to the appropriate server ensures high availability and efficient resource utilization.

This design also facilitates scaling and adapts to additional server configurations, making it a good solution for our system



# Consistent Hashing

Implemented using hashlib, a python library, it is used for distributing lists across servers in a way that minimizes reorganization when nodes lists are added or removed.

It ensures stability and efficiency in dynamic, scalable systems, making it a great tool for load balancing distributed architectures.



# Server Structure



## Web Server

Flask Web Server

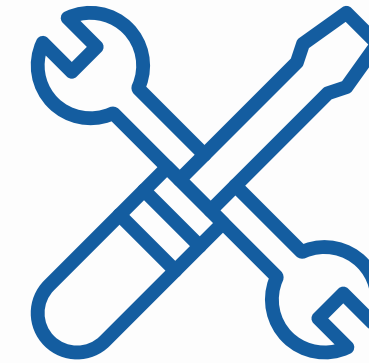
The main web server implemented using Flask, a web framework for Python. It handles HTTP requests and responses, routing, and interfaces with the underlying database to perform CRUD (Create, Read, Update, Delete) operations.



## Database

SQLite Database

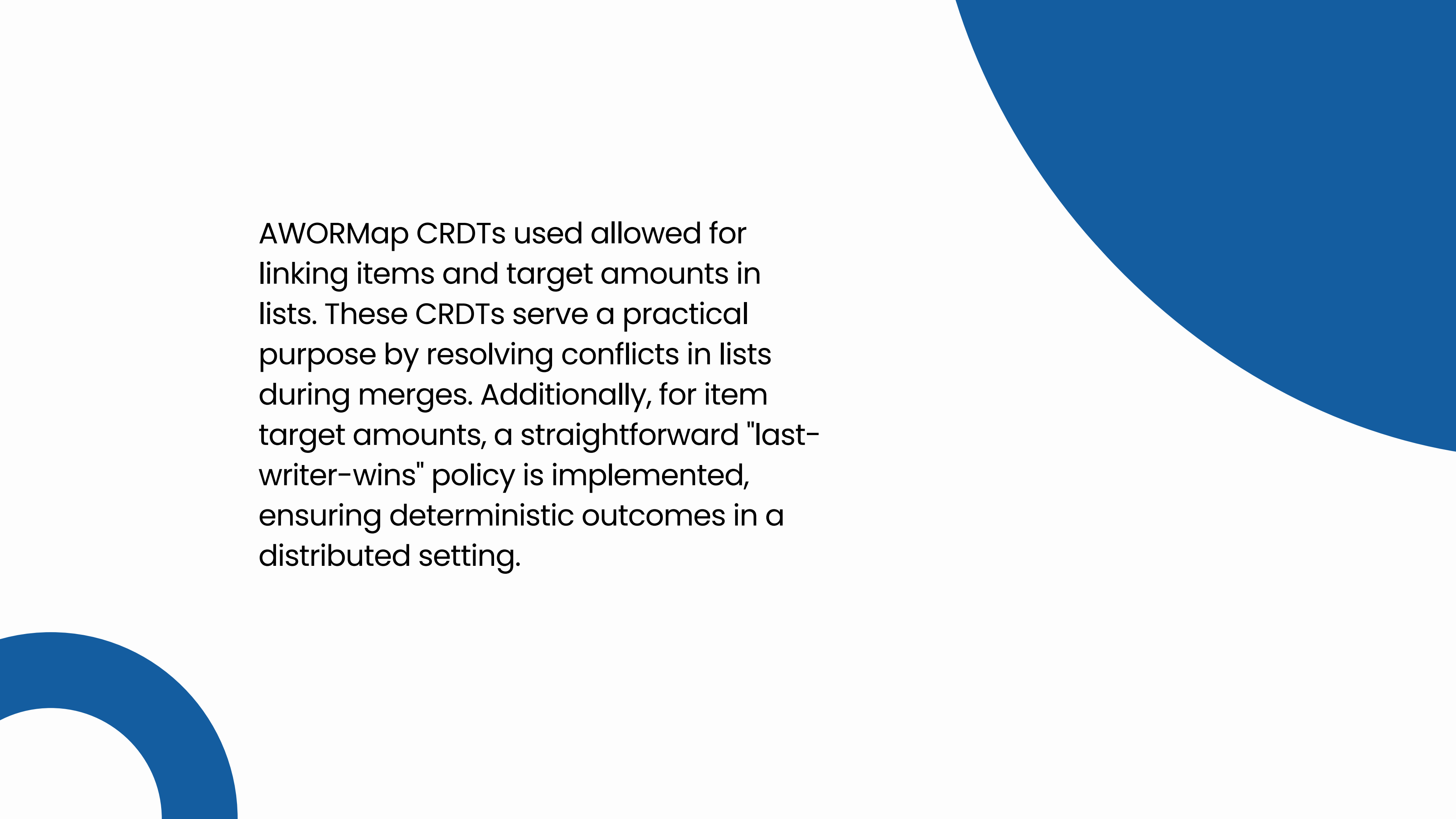
A lightweight, embedded relational database management system used to store and manage data for the server. The server interacts with the SQLite database to store and retrieve information about lists, items, and user actions.



## Action

List and Item Actions

Contains functions to perform actions related to lists and items, such as adding or updating lists, adding items to lists, and retrieving lists. The actions are defined as functions, and the server routes HTTP requests to these functions based on the requested endpoint.



AWORMap CRDTs used allowed for linking items and target amounts in lists. These CRDTs serve a practical purpose by resolving conflicts in lists during merges. Additionally, for item target amounts, a straightforward "last-writer-wins" policy is implemented, ensuring deterministic outcomes in a distributed setting.



# Major Challenges Faced

Enabling concurrent changes to shopping lists while maintaining consistency.

Balancing the local-first approach with seamless cloud collaboration.

Adapting template CRDTs to our approach.