Preface

When a disaster strikes, the main bottleneck for recovery is resources, or the lack thereof. However, this is not a problem that can be solved by simply adding more money and resources, the real problem is inefficient distribution. This is a problem that can even affect centralized organizations like NGOs and is a major prohibitor of recovery for decentralized systems. Each household possesses resources, yet faces needs. If these providers and consumers are made aware of each other, they can help each other with their deficits.

Glossary

1. Source - Person offering resources
2. Sinks - Person requesting resources
3. Nodes - Representation of Sources and Sinks algorithmically
4. Cluster - Set of one or more sources and sinks, defined geographically
5. System - Set of clusters
6. Deficits - Set of sinks which cannot be satisfied by the sources in its cluster
7. Extras - Set of sources that do not contribute to the feasibility of the cluster
8. Excesses - Set of sources which contain surplus resources after satisfaction of its cluster
9. Feasible Cluster - Subset of a cluster that has no deficits
10. Perfect Cluster - Subset of a cluster that is Feasible and minimum excess(es)
11. Free Pool - Set of Deficits and Extras in a system
12. Feasibility Score - A score that evaluates a (something) taking distance and feasibility

Functional Requirements

- Users should be able to post their needs and surpluses
- Delivery agents should be able to pick what requests they can satisfy
- There should be an algorithm to optimize requests and supplies to satisfy all requests
- The process for users to post requirements should be easy

Non-Functional Requirements

- The app should be integrated with pre-existing NGOs to help their efforts
- Measures should be implemented to prevent misuse
- There should be features to foster a sense of community among the users

- Users should be able to directly connect with other users bypassing the algorithm if they so choose

System Architecture

The core of the system is a database that users can connect to and interact with, so we implement a data-centered architecture. The database contains all the data of the needs and surpluses of all users.

Through a mobile frontend application, users can post their requirements and delivery agents can choose to satisfy requirements. In order for the algorithm to function, there must be a backend system that tracks the needs and can find optimal routes in an area. The backend is also needed for the frontend to interact with the database.

A backend is also required for integration with other organizations, it can be a public API or a specialized frontend that provides information on a region. In order to implement misuse prevention and reinforce community health, users should be able to rate other users or report them

System Requirements Specification

Users should be able to post their needs and surpluses

It should be a very simple process for a user to add a request to the system. There should also be measures in place to restrict the kinds of items that can be requested

Delivery agents should be able to pick what requests they can satisfy

It is important that delivery agents can pick how to satisfy requests themselves so that they aren’t tied to the algorithm

There should be an algorithm to optimize requests and supplies to satisfy all requests

The algorithm should have an accuracy of over 95% and take current situations into account such as inaccessible areas. It should be possible for delivery agents to override the recommendation if they feel the algorithm is not correct

The process for users to post requirements should be easy

The app should be very accessible to users so it is very important that the main functionality must be user friendly and easy to do. There should be minimal region for error

The app should be integrated with pre-existing NGOs to help their efforts

The database functions as a repository of all the requirements of all users in a region so its information is incredibly useful for the planning of centralized efforts by NGOs and can save a lot of time and manpower. Other organizations should be able to access this data and interact with the database as well

Measures should be implemented to prevent misuse

Users should be able to report other users for a variety of reasons such as not reporting their actual requirements or other misuse. The reporting system should also be factored into the algorithm to decide optimal routes

There should be features to foster a sense of community among the users

Misuse prevention starts from the user’s mindset and as such, users should be encouraged to participate in a genuine manner through positive feedback systems such as likes or reviews

Users should be able to directly connect with other users bypassing the algorithm if they so choose

The algorithm as a system can fail some users, so it should be possible for users to find people who can satisfy their requirements and directly contact them, through the app or by other means

System Evolution

The community system is very user focused and as such will change over the course of testing and throughout its lifecycle as users start exploiting the system. The direct connection system will also change during testing based on how users utilise the system.

System Models

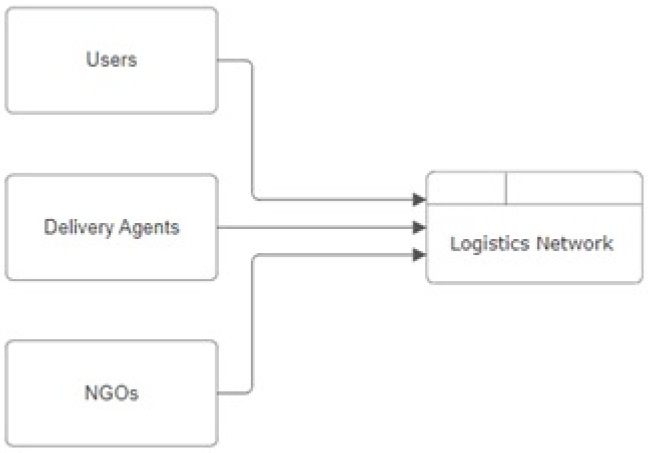
The system flow is as follows:

1. Multiple users post sets of requirements containing needs, surpluses or both
2. The algorithm uses the data posted and finds optimal paths between the users to satisfy the requirements
3. Delivery agents can find the optimal route closest to them and choose to follow the route or manually claim jobs based on their preferences
4. The delivery agents then distribute the resources and mark them as delivered, which the users must validate and the request is fulfilled

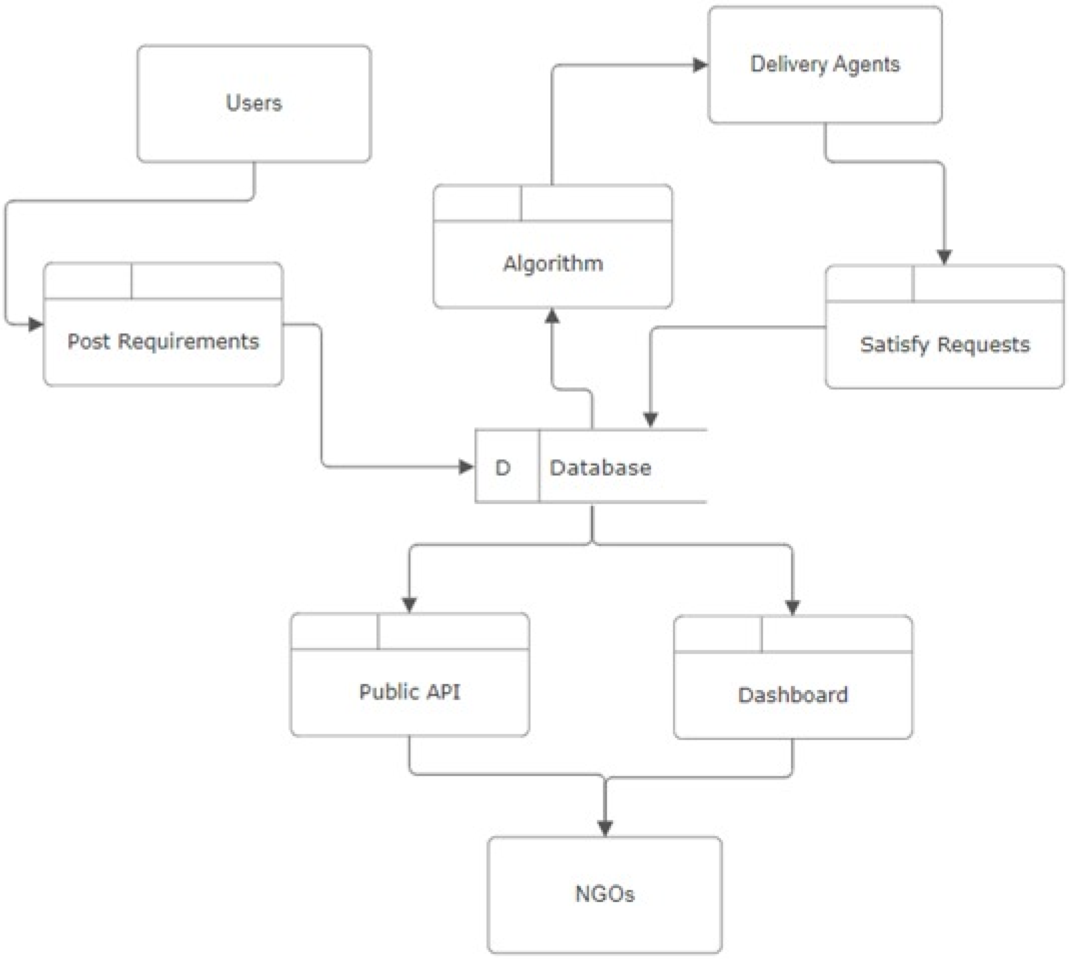
All the steps occur concurrently but every piece of data must flow in this order

Data Flow Diagram

Level 0 (Context Diagram):

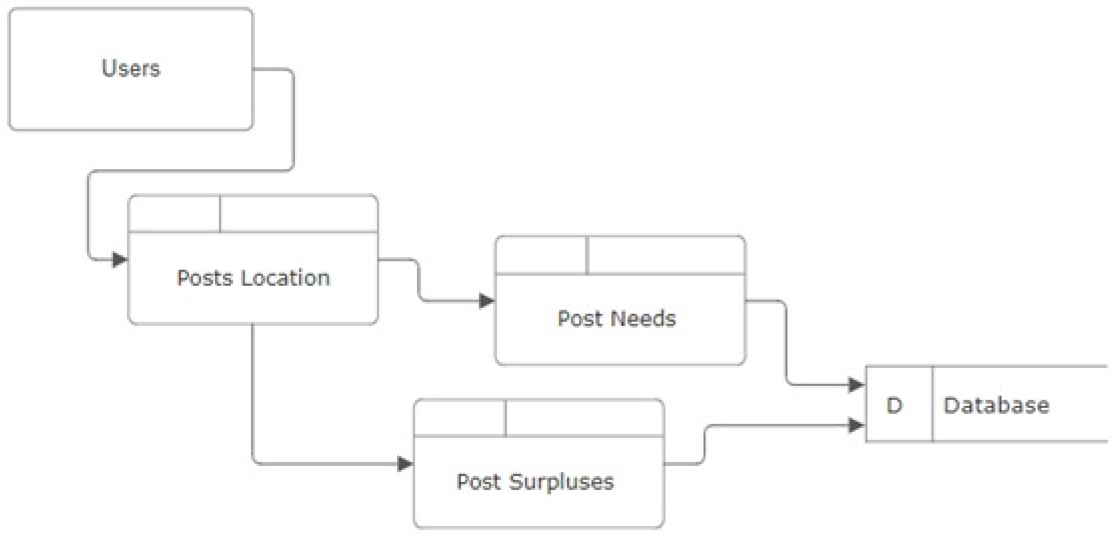


Level 1:

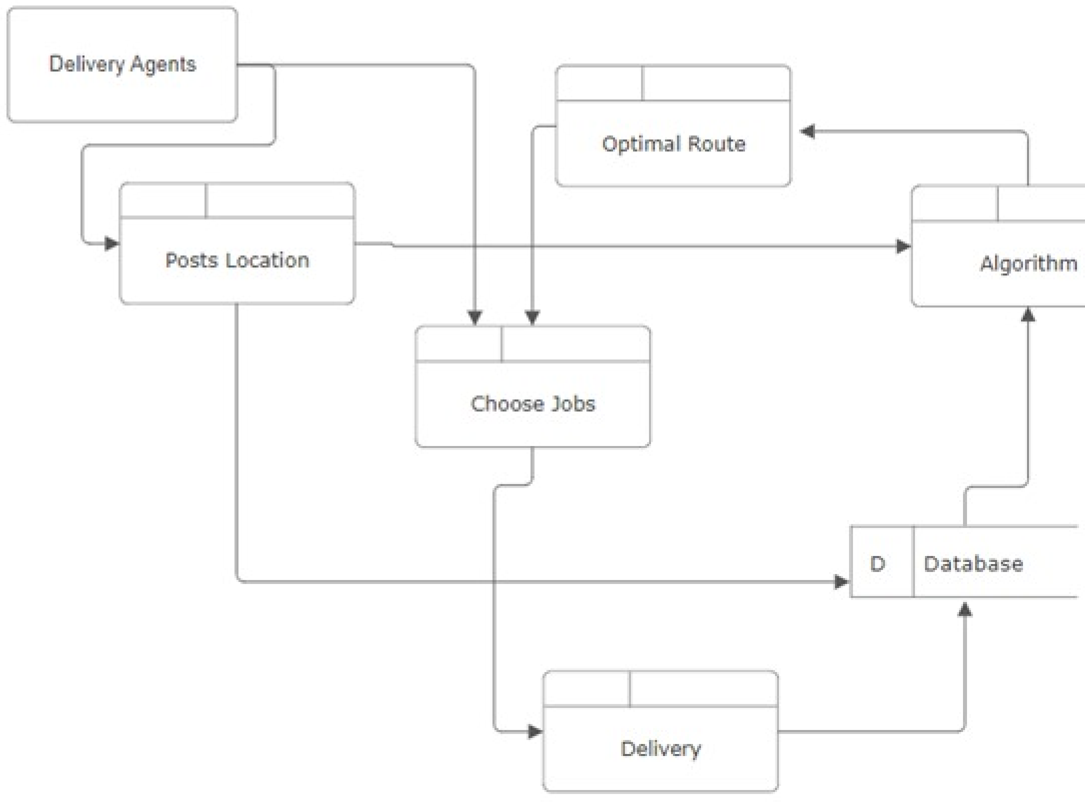


Level 2:

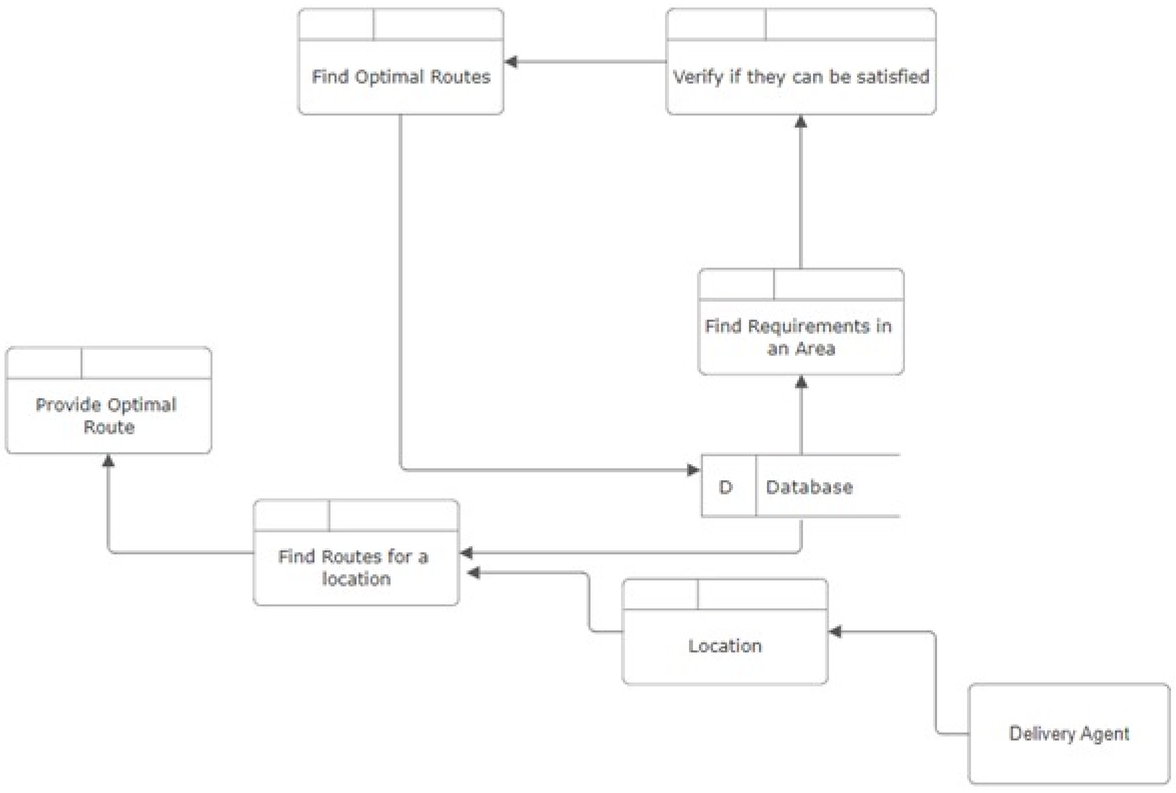
DFD for User



DFD for Delivery Agent



Level 3:


Find Optimal Routes
Verify if they can be satisfied
Find Requirements in an Area
Provide Optimal Route
D
Database
Find Routes for a location
Location
Delivery Agent