Cab Hailing API Application

PROBLEM STATEMENT

The aim is to develop a Cab Hailing API Application using FastAPI that assigns rides to customers based on their location and vehicle preference. The customer will input their present location as well as their preferred vehicle type. Given these, the closest vehicle of the preferred type must be assigned to them. The customers must be charged a fixed base fare along with additional charges based on the distance traveled. The cab details, fare, trip route, trip time and waiting time must be displayed to the customer.

REQUIREMENTS

1. The customer should be able request a ride by specifying pick up location, drop location and vehicle preference.
2. The system must allocate a cab of the preferred vehicle type nearest to the pick up location.
3. The fare must be calculated with fixed base fare along with additional charges based on distance travel.
4. The fare must also vary based on the vehicle type of the ride.
5. The customers must be allocated equitably between the cabs so that all drivers can take a fare share of trips per day.
6. The application must update the cab data by incrementing the trip count and changing the location of the allocated cab to drop location.
7. The cab details (vehicle type, driver name, license plate no.), fare, waiting time, trip time and shortest trip route must be displayed to the customer.
8. The application must be scalable to a change in the number of nodes (in the matrix of the graph to represent the map) or a change in the number of drivers.

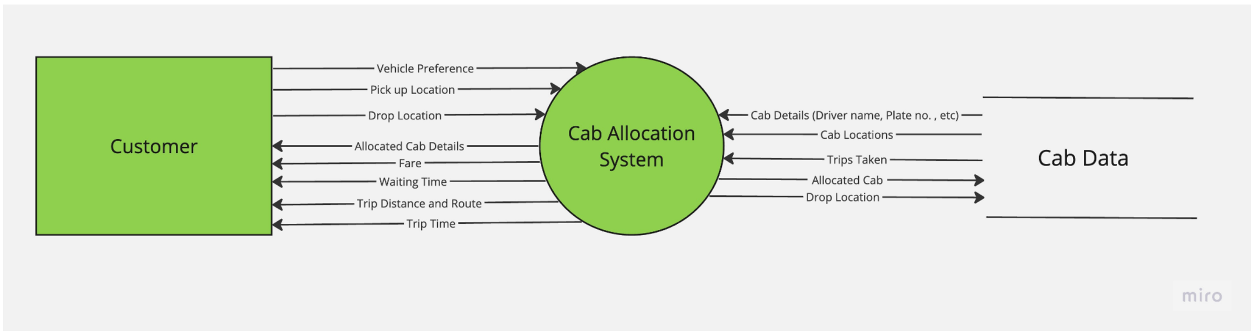
CONSTRAINTS

1. The number of drivers is fixed
2. The number of graph nodes is fixed.

3. Some graph nodes are inaccessible.

ANALYSIS USING DATA FLOW DIAGRAM

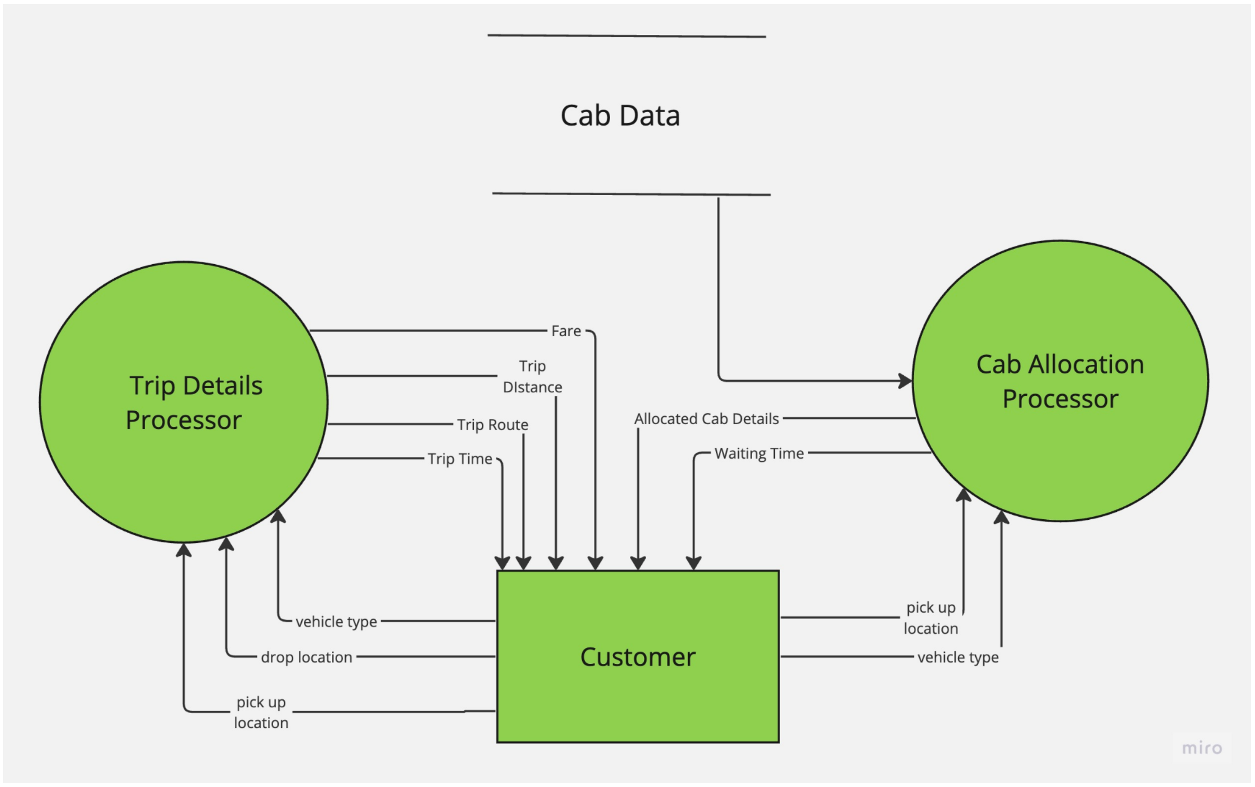
Level 0 Diagram



As a starting point, the above Level 0 diagram represents the flow of data between the customer (user), the cab hailing application and the cab data file.

The cab hailing application takes vehicle preference, pick up location, drop location as input from the customer. It fetches the cab locations, trip count and cab details from a file. It then allocates a cab as per requirement. It returns the details of the allocated cab and drop location to update the location and trip count in the cab data file. Finally, the allocated cab details, fare, waiting time, trip distance and route and trip time is displayed to the customer.

Level 1 Diagram

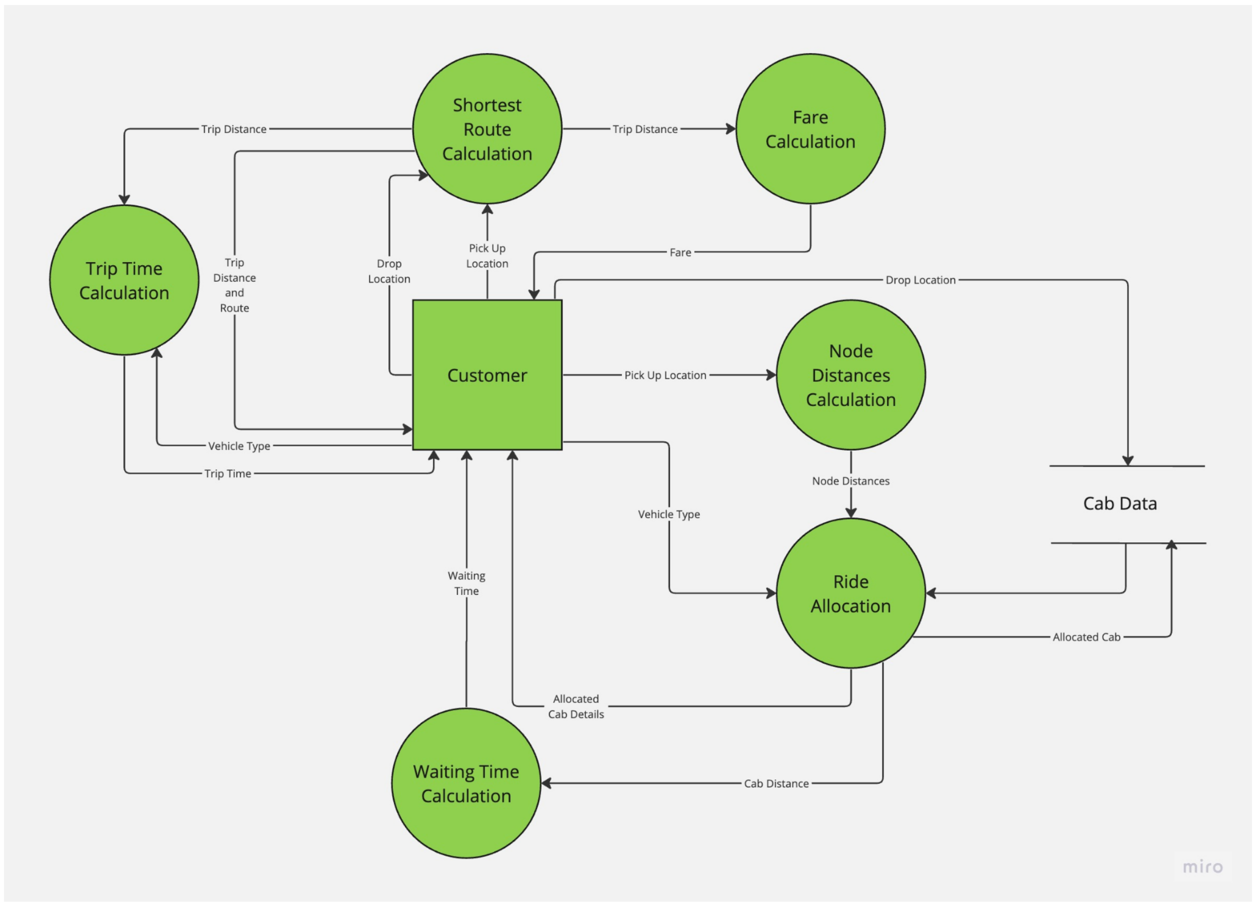


In the Level 1 DFD, two processes are identified: Cab Details processing and Trip Details processing.

The Cab Allocation Processor takes pick up location and vehicle type from the customer and returns back the allocated cab details and waiting time.

The Trip Details Processor takes vehicle type, pick up location and drop location from the customer and returns the fare, trip distance, trip route and trip time to the customer.

Level 2 Diagram



The above level one diagram gives an overview of the flow of data between the different modules.

First, in the Cab Allocation processes, the pick up location from the customer is taken into the Node Distances Calculation Module, which gives the distance of the pick up location to all other nodes in the map using Djikstra’s algorithm. These distances are passed to the Ride Allocation

Module along with the vehicle preference of the customer and data from the Cab Data file. Once the driver is allocated, the cab data file is updated for trip count and location of the cab allocated. The details of the cab are displayed to the customer. The cab distance is sent to the Waiting Time Calculation Module to calculate the waiting time and display it to the customer.

In the Trip Details processes, the pick up location and drop location from the customer is passed to the Shortest Route Calculation Module, which calculates the optimal trip route using Djikstra's algorithm. This module returns the trip route to the customer and passes the trip distance to Fare Calculation Module (which also takes the vehicle preference) and Trip Time Calculation Module which return the fare and travel time of the trip to the customer respectively.

CAB DATA FILE

The cab data file contains the information on the available cabs. The information is stored in a tabular form. Each entry contains:

- a unique cab ID
- driver name
- license plate no.
- vehicle type
- location
- trip count

VARIABLE DESCRIPTION

Variable name: locations

Type: dictionary

Description: a dictionary of location names and the node number representing them on the adjacency matrix that maps the place. The format is as follows:

- Location1 : its node number on the matrix
- Location2: its node number on the matrix
- . . .

Variable name: dist_path

Type: dictionary

Description: A nested dictionary of locations and the shortest distances and path to them:

- node1 : {“distance”: shortest distance to it, “path” : list of nodes that describe the path}
- node2 : {“distance”: shortest distance to it, “path” : list of nodes that describe the path}
- . . .

Variable name: adj_matrix

Type: nested list

Description: A nested list that represents an adjacency matrix. Say each element in the matrix is represented by row i and column j, where i and j represent nodes. Then it stores the distance between i and j if they are adjacent nodes, and 0 otherwise.

Variable name: allocated_cab

Type: dictionary

Description: A dictionary consisting of cab details formatted as follows:

- “driver” : name of the driver
- “license plate. “ : license plate number
- “vehicle type” : the type of the vehicle
- “location” : the present location of the vehicle
- “distance” : the shortest distance between the vehicle and the pick up location

Variable name: vehicle_fare_rate

Type: dictionary

Description: A dictionary of the vehicle type and its fare rate formatted as follows:

- “Vehicletype1” : fare rate of the vehicle type
- “Vehicletype2” : fare rate of the vehicle type
- . . .

FUNCTION DESCRIPTION

Function name: location_to_node

Description: Converts the location name to its corresponding node number on the adjacency matrix

Parameters: locations dictionary, location name

Returns: node number

Function name: node_to_location

Description: Converts the node number its corresponding location name

Parameters: locations dictionary, node number

Returns: location name

Function name: user_input

Description: Accepts inputs from the user

Parameters: None

Returns: A dictionary of user input:

- “Pick up” : pick up location
- “Drop”: drop location
- “Vehicle”: vehicle preference

Function name: shortest_distance

Description: Accepts pick up location and calculates its shortest distance and path to every other node or location

Parameters: pickup node, adj_matrix

Returns: dist_path dictionary

Function name: allocation

Description: Allocates the cab at the shortest distance of the vehicle type specified. If two such cabs exist, then the cab with the lower trip count is allocated.

Parameters: cab data file, dist_path dictionary , vehicle preference

Returns: allocated_cab dictionary

Function name: trip_distance

Description: Finds the trip distance by getting the shortest distance assigned to the drop location in the dist_path dictionary

Parameters: dist_path dictionary, drop location

Returns: trip distance

Function name: time

Description: Calculates time taken to cover a distance

Parameters: distance, speed

Returns: time

Function name: fare

Description: Calculates the fare based on based on vehicle type, base charges and additional charges

Parameters: vehicle_fare_rate dictionary, base charges, trip distance

Returns: total fare

Function name: display_details

Description: Displays the allocated cab details, i.e., driver name, license plate no., vehicle type, cab location, waiting time, travel time, trip route, trip distance, travel time and total fare.

Parameters: allocated_cab dictionary

Returns: None