SheCodes Hackathon

Team Name: Semicolon

College Name: National Institute of Technology, Srinagar

Team Leader’s Name: Ruchi Khandelwal

Team Members

Member's Name	College Name	Email Id	Roll No.
Ruchi Khandelwal	NIT Srinagar	ruchikhandelwal914@gmail.com	2021BITE039
Disha Baghel	NIT Srinagar	bagheldisha708@gmail.com	2021BCSE049
Snigdha Mahajan	NIT Srinagar	snigdhamahajan15@gmail.com	2021BITE026
Diya Puri	NIT Srinagar	diya.puri2912@gmail.com	2021BITE022
Akshita Singh	NIT Srinagar	akshitasingh202@gmail.com	2021BCSE017

Topic: Matching customer with fleet

Through the development of an algorithm and database schema that effectively matches clients with fleet agents for order fulfilment, this project seeks to improve Airtel's service delivery process. Making sure clients receive the services they've asked as soon as precisely and swiftly as possible is the goal.

The decision to pursue this initiative is indicative of a dedication to improving operational effectiveness and customer service within Airtel's service delivery framework. Our goal is to simplify the order fulfilment process by refining the method of linking customers with fleet agents according to their service requirements and geographic areas. For each and every Airtel customer, we aim to provide a flawless and customised experience.

Solution approach

Define problem

Diverse services are provided by Airtel customer care, which also strives to enhance offline services. In order to do this, agents must be sent to customers' locations in response to service requests. But for a big client base, scalability, accuracy, and efficiency are essential factors. Here, the issue will be divided into two smaller issues:

- Create an algorithm that can accurately, efficiently, and scalable match the agent's position to that of the client.
- Create a database schema or model that works well for storing data and creating queries that match data to retrieve information.

Research and Understanding

In order to better comprehend and research the topic, we have visited airtel's website, many other agents that provide services, the airtel customer care number, and other sources to learn more about doorstep services. This will enable us to address the issue in real time. We learned about a number of services, including internet, DTH, sim cards, and networks. We have included a entity relation diagram fig 1 for this:

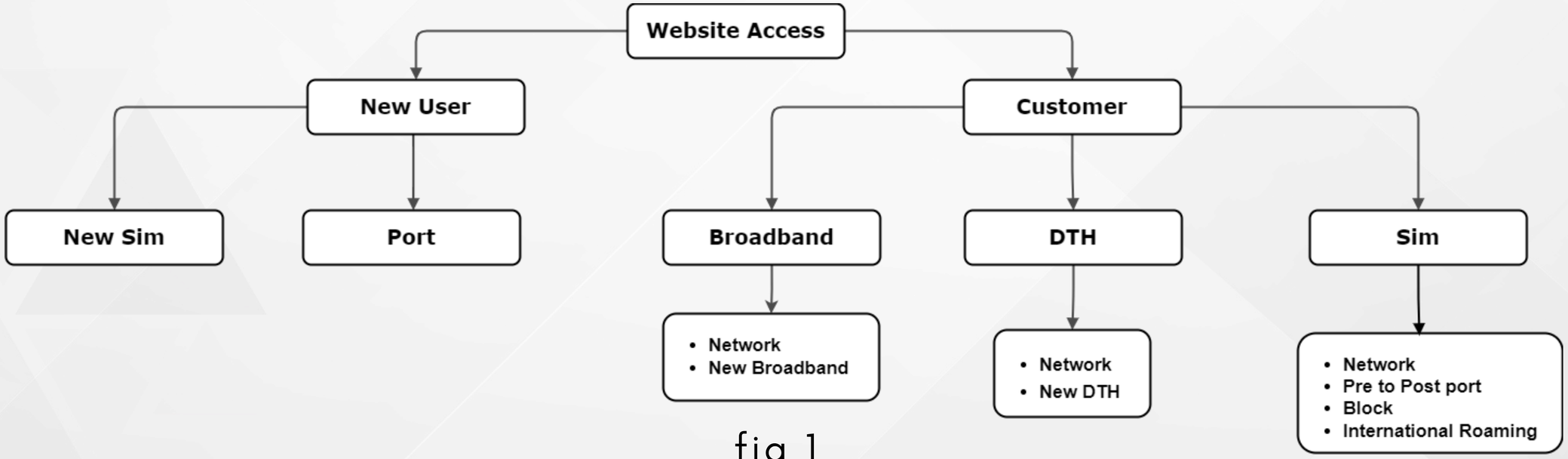

fig 1

TECHGIG

This led us to decide to split this project into two sections: the database and the algorithm. We conducted independent study on each of them, as detailed below:

- In terms of algorithms, we read through numerous research publications as well as a number of algorithms, including Dijikstra, BFS, DFS, and A* which helps in extracting the shortest time path.
- In addition, we investigated other formulas and APIs, such as the Haversine and Euclidean distances, and created an ideal algorithm.
- Since the database model affects the entire process, we have employed PostgreSQL to better understand it.
- The database schema effectively stores information for agent-customer matching. Separate tables for different aspects (customers, agents, assignments).
- Using tools like psycopg2 and sqlalchemy, we can successfully link databases and algorithms to produce an agent-like result.

Requirement Analysis

After comprehending the issue, we examined the necessary conditions to be:

- End users' requirements and preferences, such as those of customers and agents, in order to guarantee that the system fulfils their demands and improves their experience.
- Establish protocols for receiving and handling consumer requests via different channels including apps, web, customer support, and in-store visits.
- Establish the standards for assigning agents to client requests, taking into account variables such agent expertise, distance from the customer, availability, and workload.
- To enable prompt matching and order fulfilment, indicate that real-time information on agent availability and customer locations are required.
- Determine the level of scalability needed to handle a high amount of client requests and agent assignments, and establish performance standards for system throughput and response times.
- Define requirements for data storage, backup, and access control to ensure the security and integrity of customer and agent data.

Solution Design

The solution design can be approached in various ways, such as brute force, optimal, or other methods. In this project, we chose to implement the optimal approach. Initially, we divided the problem into two main parts:

The Algorithm Design:

Entity relationship diagram fig 2 and it's explanation.

- allocate_agent function:

1. Initialize a minimum distance threshold (min_distance) to ensure that only agents within a certain distance from the customer are considered.
2. Filter the list of available agents based on their status (available) and haversine_distance from the customer location. Adjust the minimum distance threshold dynamically if no agents are found within the initial threshold.
3. Initialize a graph representing the geographic area using data obtained from a map API. Each node in the graph represents a location, and edges represent travel time or distance between locations.

4.Use the A* algorithm to find the shortest time path from the customer location to each available agent's location. Incorporate a heuristic function based on Haversine distance for estimating the time to reach each agent's location.

5. Select the agent located at the goal node of the shortest time path as the allocated agent for the customer.

6. Return the allocated agent.

- get_haversine_distance function: This function calculates the distance between two geographic coordinates using the Haversine formula. Ensure that this function accurately calculates distances between agent and customer locations.

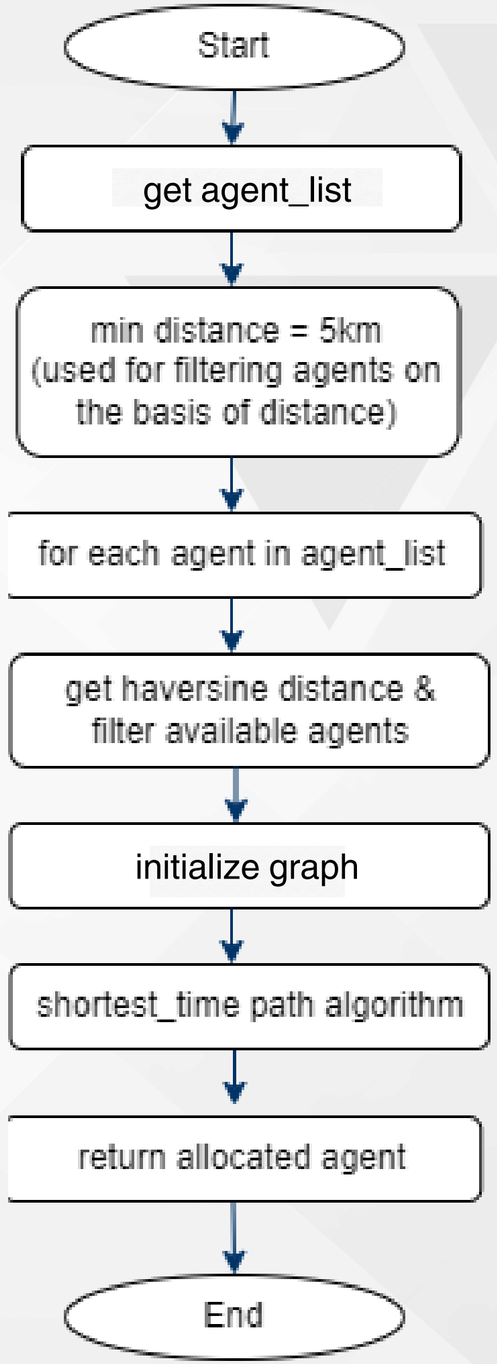

fig 2

Implementation

TECHGIG

Pseudo code for algorithm

```
function get_haversine_distance(agent_location);

function allocate_agent(agent_list):
    min_distance = 5km

    //filter agents on the basis of available status and, haversine distance.
    list available_agent
    for each agent in agent_list:
        haversine_distance = get_haversine_distance(agent_location)
        if status == 'available' and haversine_distance < min_distance:
            available_agent.append(agent)
    if (available_agent == [])
        min_distance += 2  then repeat the filtering steps again
    //initialize the graph using data from any map api
    //geocoding api and distancematrix api
    setgraph(use time_extracted_using_api for edges)

   //A* algorithm to find path the best agent who can reach to customer
in minimun time_extracted_using_api
    starting_node = customer
    heuristic_function = haversine_distance__between_two_nodes.
    g = edge_of_graph i.e., time between each node
    f = g + heuristic_function

    calculate_shortest_time_path_using_AStar_algo()
    allocated_agent = goal_node_from_shortest_path
    return allocated_agent
```

Formula for haversine distance

$$haversine(\theta) = sin^2\left(\frac{\theta}{2}\right)$$

$$\left(\frac{d}{r}\right) = haversine(\Phi_2 - \Phi_1) + cos(\Phi_1)cos(\Phi_2)haversine(\lambda_2 - \lambda_1)$$

$$d = rhav^{-1}(h) = 2rsin^{-1}(\sqrt{h})$$

or

$$d = 2rsin^{-1}\left(\sqrt{sin^2\left(\frac{\Phi_2-\Phi_1}{2}\right) + cos(\Phi_1)cos(\Phi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

The Database Design:

TECHGIG

Entity Relationship Diagram

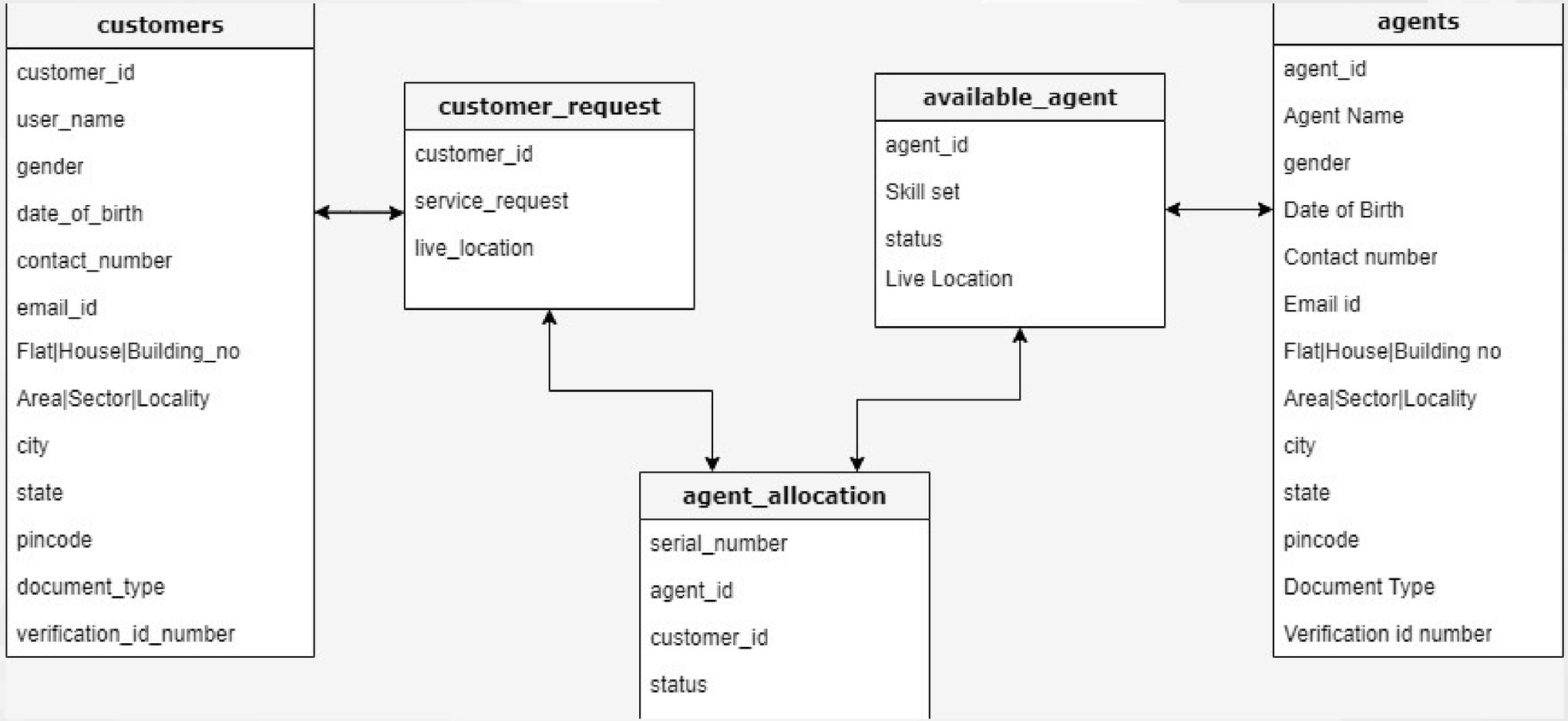

fig 3

Key Points

TECHGIG

- An ERD illustrates entity relationships and attributes.
- 'customer_request' references 'customers' via 'customer_id'.
- 'available_agent' references 'agent' via 'agent_id'.
- 'agent_allocation' links 'customer_request' and 'available_agent', assigning available agents to service requests.
- Following the extraction of relevant data, an algorithm matches the best-suited agent to the customer's request, resulting in an entry in the 'agent_allocation' table and updating the status of the assigned agent to 'busy'.
- Upon completion of the requested service, the 'status' column in the 'agent_allocation' table is reset to 'Available'. Simultaneously, the 'status' column in the 'available_agent' table is updated to 'Available' using the 'agent_id' key. Additionally, the corresponding record in the 'customer_request' table, identified by the shared 'customer_id', is removed.
- Finally, after the necessary updates and deletions, the entry in the 'agent_allocation' table pertaining to the completed service is deleted.

Integration of algorithm and database

TECHGIG

- Utilize Psycopg2's to map database tables and relationships, and optimize queries.
- Manage transactions with Psycopg2's session management, ensuring data consistency, and enforce integrity through indexes, constraints, and validation features.
- Use Psycopg2's migration tools for seamless schema updates and configure connection pooling settings for efficient database connections management.

Query for information retrieval

```
query = """
SELECT cr.customer_id,
aa.agent_id, cr.live_location AS "Customer
Location", aa."Live Location" AS "Agent Location"
FROM customer_request cr
JOIN available_agent aa ON cr.service_request = ANY(aa."Skill set")
WHERE aa.status = 'Available';
"""
```

Scalability

- Horizontal Scaling: Splitting data into smaller parts and storing each part on different servers for better scalability.
- Database Normalization: Organizing data in a database to reduce redundancy and improve data integrity, leading to better query performance and reduced storage needs.
- Query Optimization: Improving PostgreSQL query performance through techniques like indexing, query rewriting, and data caching.
- Distributed Database Systems: Storing data across multiple servers for enhanced scalability and efficiency compared to traditional relational databases.
- Implement caching for optimal routes between common locations to save processing time.
- Use an asynchronous task queue for concurrent task processing in large-scale deployments.
- Optimize distance filtering with spatial indexing schemes for faster agent retrieval.
- Explore scalable pathfinding algorithms for efficient routing in extensive networks.

Technology Stack & Links

Python PostgreSQL GoogleMapsApi

SQLAlchemy Psycopg2

Github Links for the database model:

https://github.com/SnigdhaMahajan/Blue

References:

https://onlinepubs.trb.org/Onlinepubs/trr/1993/1408/1408-012.pdf

https://www.geeksforgeeks.org/shortest-path-algorithms-a-complete-guide/

Screenshot of result of page 12 query:

```
Customer ID     Agent ID        Customer Location    Agent Location
-------------------------------------------------------------------
1               102             (40.7128,-74.006)    (34.0522,-118.2437)
2               101             (34.0522,-118.2437)  (37.7749,-122.4194)
2               102             (34.0522,-118.2437)  (34.0522,-118.2437)
3               101             (41.8781,-87.6298)   (37.7749,-122.4194)

Process finished with exit code 0
```

Solution to GPS Accuracy

To ensure precise location accuracy within a range of 10 to 30 meters, particularly in indoor environments where GPS signals may be attenuated, meticulous measures could be undertaken

- By using correct and accurate distance/time apis
- Feedback Mechanisms
- Wifi Positioning Systems
- Communication with customer to get its exact location

TECHGIG

Thank you!!