Main challenge is to virtualize the Whole public transport System, how do we do it?

Thousands of Roads in a city, hundreds of hubs/Chowks, how do we make each Route Unique, on top of that each hub is connected with multiple roads, each road being bidirectional.

Let's establish the main entities involved in a public transport System:

1)Hubs/Chowks:- An open market area in a city at the junction of two roads ,Where public transport vehicles queue up to load passengers.

2)Routes/Roads:- The interconnecting route between the two hubs.

3) Public transport Vehicle and Drivers :-Buses and Rickshaws drivers which traverses on a fixed repeated path

4)Passengers:- Looking for a ride from a public transport vehicle.

All these need to have a unique Id/ value to distinguish them from any other and also to stablish relations between them.

Unique Id/value for entities 3 and 4:

1. The Public Transport Vehicles can be divided into two Major types (3):
   - The driver is the primary Entity(Auto- Rickshaws /E-Rickshaws): Auto/E- Rickshaws can change but the driver remains the sole entity driving on the Fixed path. After OAuth-based sign-up, the system assigns a unique ID to the user as driverId.
   - The Vehicle is the primary entity(Buses):Drivers can change but the same Bus runs on the fixed Route. Unique id will be associated with the IOT device,
     [Refer ./ TransitSync_Architecture&Code/System_Design/Future_System_design_PNG
     OR
     https://drive.google.com/file/d/1xZFluCYvFOCfxDhqju4HqR9Q1MTrtAyV/view?usp=drive_link]
     This IOT device will be attached to the Bus.

   Entities will continuously Update their Location to the key Unique Id for passengers to track it.

2. The Passengers (4) :After OAuth-based sign-up, the system assigns a unique ID to the user as PassengerId.

Now, let's start with Rickshaws, as we know drivers queue about the chowks with a rule of first come first serve amongst the drivers, first one gets to load up all the passengers and then the driver leaves, following which second in queue now gets to load up passengers as now being first in the queue.

Same chowk can have multiple roads branching out of it, let's assume a 4-way chowk, simultaneously there are 4 drivers that are at queue position 1. Now let's take a more complicated case , one road which forks later and drivers have separate queue picking up passengers for their route , now the

number of drivers at queue position 1 itself at the chowk is 5 (3 for non-forking roads and 2 for forking road) simply queuing a driver to let's say Rajiv chowk won't work, we need uniqueness.

Let's start with hubs/chowks : Fortunately, this challenge has been tackled before , map Systems such as google maps have already classified chowks/hubs or places with even slightest importance regionally giving them a unqiue Id called places ID.

Example of Places Id:- ChIJgUbS6P_EyzQRQYy2Ld1U5wA

Note: Information about something Unique can be very well Stored in key values Pair, especially for fast access such as from Redis.

Now that we have a unique Id for a hub we can Store all kinds of data related to it in a value hashSet/dictionary about the Chowks exact latitude & Longitude, other hubs connected to it, Routes spanning out of it or into it, City in which the hub is located etc.

Now that we have all the hubs mapped and virtualized, we need find unique Id for each route between two hubs.

Process of creating Unique ids for each Route:

Route Between two points/ Hubs are made up of Latitude and Longitudes defining the Segment of road between the two hubs.

Efficient way to store latitude and longitude data for a route is using Polyline encoding format. [Refer to Project report appendix polyline] but this String can be of variable length. For example having a 500 character Key for storing related data to the Route wouldn't be logical. [Note: Unique ids of entity will always be used as key for storing essential Values about that entity]. To generate a consistent unique identifier for a route, we hash the polyline string using SHA256.. This is used for storing Values about the unique routes for example Fare(price), distance, original Route Points(polyLines), start Hub and end Hub etc.

Example of SHA256 unique Id of Routes: 272010d7bc6d5a3264b77aa2f7fe2c30

Now we have a unique Id for Each route and each Hub, now we can link all Hubs with each other based on the connecting unique Routes and Vice a versa.

Two Hubs can have a single route or Multiple Routes connecting them

Lets take an example StartHub: Rajiv Chowk[PalcesId: ChIJwZPwLMrpRDcRU-9ghbWQ_Tg] and EndHub: Pranabh Kunj[PalcesId:ChIJEUQx_cjrRDcRO2EVVTwOpig]

 To stablish all relation between them lets make unique identifier for relation between two hubs, this can simply be PlacesId of StartHub:PalcesId of endHub ->

ChIJwZPwLMrpRDcRU-9ghbWQ_Tg:ChIJEUQx_cjrRDcRO2EVVTwOpig

One can store all related data between the two connected hubs in this Unique key , I Currently only Strore the all the PolyLines/Route Paths between the two hubs.(Two hubs can have multiple routes connecting them)

Now that we have a Unique hub to hub relation and Unique Route connecting them we can Finally Queue the drivers on the specific Route.

Example Key->[driverId1,driverId2…drverIdX]

The driver Could just continuously update Its current Lat and Lng/position to the key driverId(Generated During driver Signup) and the passenger on the route could just fetch the driverId first in queue to track.

Only hurdle being Bi directional Roads, we tackle this by simply creating a key for stationary Queue

->startHubId:routeId

Example->ChIJwZPwLMrpRDcRU-9ghbWQ_Tg: 272010d7bc6d5a3264b77aa2f7fe2c30

to which drivers Will be queued. On Starting the traversal/ride the driverId will be removed from stationary Queue to In transit list. With the key-> startHubId:routeId:InTransit->

Example->ChIJwZPwLMrpRDcRU-9ghbWQ_Tg: 272010d7bc6d5a3264b77aa2f7fe2c30:InTransit

Similarly the passengers will also be added to the key-> startHubId:routeId:Passenger

Example:- ChIJwZPwLMrpRDcRU-9ghbWQ_Tg: 272010d7bc6d5a3264b77aa2f7fe2c30:Passenger

To which active passengers will be added [Unactive will be popped] this will help in maintaining live passenger Count for driver to see.

[Whole Process is well explained Refer to the Project Report(9.2.2 Insertion of Driver and Passenger in Registered Route)]

[Note: Both Driver and passenger are Bound within the bounds of the Route using bounds Builder Containment]

FlowChart- https://drive.google.com/file/d/1App1XzW6yjps5vzg5jECOtKT9rrcenB_/view?usp=drive_link

Simple Understanding of Request Response-

Now that we have Understood the standardization and Virtualization of Hubs and Routes, we will discuss the work Flow:

[Refer to ./Project Report 9.3.3]

Every Thing is dynamic we only provide the architecture:

First the driver Registers to a route that he/she Generally traverses:-

Route Registration[Refer ./Project Report(9.2.1 Route Registration Feature).

A more complex Version of Route Registration is implemented in ComplexCreateRoute.java which allows creating complex Route by joining intermediate hubs between Start and End Hub.

[Refer TransitSync_Architecture&Code/transitSyncDriver/app/src/main/java/com/example/transitSyncDriver/ ComplexCreateRoute.java and ./FieldTest&Ui/ComplexRouteSelect OR https://drive.google.com/file/d/1l2jTDdY-9eMvH6xFVcbDi1o6lJC19mmD/view?usp=drive_link]

After the Route Registration the driver can select from all previously Registered Route in Route Select Activity [Refer ./FiledTest&UI/DriverEnd_Application_TransitSyncDriver(Route Select Activity) OR

https://drive.google.com/file/d/1M37dWTuhZImAdlN90JSpcFo9Sr9PziUL/view?usp=drive_link]

The selected Route is the one driver desires to traverse that day, The driver Should reach near Any one of the hubs to be Queued to it. (Lets say 100 meters of the Hub Lat Lng)

[Refer ./FiledTest&UI/Field_Test_Route_Registration_and_Queuing OR https://drive.google.com/file/d/1F5IS4xDu6z9gN25CGxwsAilFgGztfCJl/view?usp=drive_link]

The passenger on the route is required to select the Start and End Hub . They first will receive all drivers in Transit i.e one that left the stationary queue and are traveling in the Route.

[Refer ./Project Report 9.2.2.1 Driver Trip Lifecycle and Queue Management Workflow]

The passenger end of application will track the driver in Transit Drivers First , then if there are no in Transit drivers, on Request will fetch the driverId First in Queue for tracking.

Note: The drivers second , third etc in stationary queue Won't just sit Still , Lets say, a person want to reserve a Auto form his Home to a destination then ,GeoSeacrh for the nearest Hub and then the

driver second ,third etc will be sent a notification to accept the ride.Effectively removing congestion at the Hubs and also providing more opportunity for the drivers.

# For Future Implementation for buses:

The Route Registration can be Hosted on the web which will be available on scan of the QR attached to the IOT by the governing authority. Initial Route can be created just like for Auto/E-rickshaws , When the driver First traverses the route they can press the button on IOT device at each bus stand , which will register the bus stops on the Route , this will effectively stablish the Route and bus stops for the Bus entity.

Since now we have a data of the bus stops , the logged in passenger after scaning the same QR will be able to book tickets from one bus Stop to other. Through Online payment mechanisam.

Every time a Person buys a Ticket , they will be added to the Passenger count inside the bus. We have this information as well as the location of the bus through IOT, So any potential passenger on a further Bus stand can track the bus and also see the number of passengers inside . If , the passengers sees the bus is full considering their time constraint could choose to book a cab or even reserve a Auto/E-rickshaw from the nearest of Hubs from it.

Intelligent Routing:

Interestingly every Time a hub is added To the database with the interconnecting Route, it forms a network of nodes.
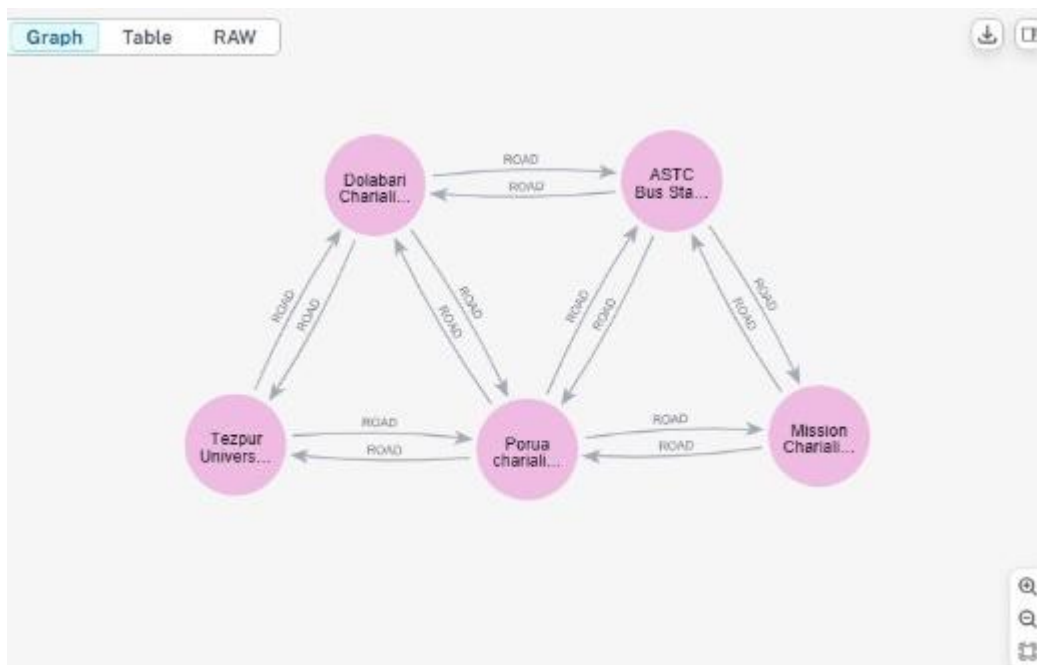
Now, Dijkstra's algorithm can be used on this hub-to-hub graph to compute the shortest path from One Hub to another.

[Refer ./Project Report (10. The Intelligent Routing System) & ./FieldTest&UI/Intelligent_Routing_UI

OR https://drive.google.com/file/d/1tDdi_mzo4xSGpZhm-xDnPFpUstvvOOIH/view?usp=drive_link]

1) Tezpur University, Napaam, Tezpur, Assam, India → 2) Dolabari Chariali, National Highway, 37A, Dolabari, Tezpur, Assam, India → 3) ASTC Bus Stand Tezpur, Mahabhairab, Tezpur, Assam, India

1) 26.7002961,92.83076009999999 → 2) 26.6516183,92.8231953 → 3) 26.620921,92.7945543

Price: 60.00
Distance: 14.60 km

So, a person can Go To any City and search the best possible route to take, with total Price.

All this Still only presents the prototype. Far more needs to implemented and considered before deploying this System in real World.

Basic functionalities to be implemented: -

1)Driver Registration and signup: The Driver registration signup etc are not implemented, hardcoded values such as "driver001" etc,


```
private String startCity="";   6 usages
private String driverId="driver001";   2 u
private String routeId="";   2 usages
```

Important-

A proper functionality not just a o-auth based Signup System but a driver verification through driver Licence and other registration thresholds for a driver should be followed.

2)Passenger singup: Currently the Code uses a hardcoded Id lets say "passengerId1"


```
// ScheduledExecutorService for periodic execution
ScheduledExecutorService scheduler = Executors.newScheduledThreadPool( corePoolSize: 1);
scheduler.scheduleWithFixedDelay(()->{
    CountMessage countMessage=new CountMessage( active: true, id: "passengerId1", routeId_startHub_passenger: routeId+":"+startHubId+":passenger");
    Call<CountMessage> call=apiService1.sendCount(countMessage);
    call.enqueue(new Callback<CountMessage>() {   ± kumar-yogesh08
```

The passenger signUp can be simple O-Auth based System.

3)Route Registration for buses and bus stop insertion on the route , currently the project hasn't practical implemented IOT device system for buses.

4)Use of open Streat maps , to reduce Api calls. Can also be used in combination with googles Directions Api for finding out alternative routes other than provided by Google.

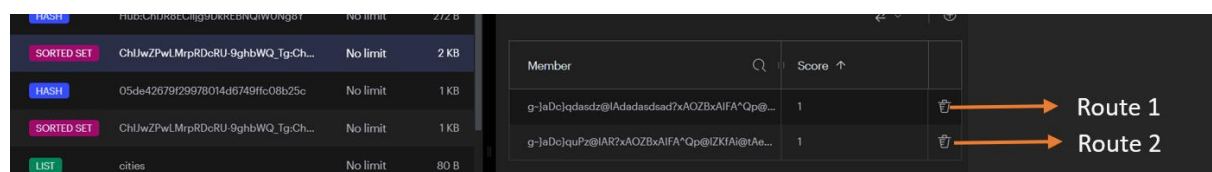## Consideration for future implementation: -

Currently in route Registration the Code forces the driver to a already existing hub if driver input-ed hub is within 400 meters of any other hub in the database, as No two chowks/hubs can lie within 400

meters. This Effectively makes the hubs and queuing uniform, also prevents cluttering of hubs by dynamic updates by driver route Registration.

We also need to consider the same for the polyline and routeId connecting the two hubs , so as to prevent redundant addition of paths if the paths have only a little bit of difference between them, generally the polyLines given by google are consistent, but considering a edge case expecting a change, what we can do is when ever a route is posted by the driver for route Creation ,

```
// Make a post Request for Route Creation
postData(encodedSelectedPath, price, email: "transitDriver1@gmail.com", driverId, vehicalNO: "V0020",
        startHubId,endHubId,startHubName,endHubName,origin,destination,startCity,endCity,distanceBtwHub);
```

the post request will Contain   startHub and EndHub, get back all the routes stored in startHub:EndHub (Explained Above)

After You get the PolyLines decode them and compare them using Lat Lng builder , lets say there are N Points in one polyline example-Route 1, then there will be N/2 bounds builder between each point on the route and its successive points i.e. a Lat Lng builder made out of just two points. Compare each point of post request decoded PolyLine to Route1 to RouteN from startHub:EndHub.

As, a point can shift if google polyline algorithm changes but the area created by the point and its successor will most likely contain the new point from the changed algorithm. This comparison will be computationally expensive so use a micro-Service(docker) for this which could independently process this while you deal with other processes. If 90 percent of the path matches prompt the driver that route already exists and please Go to select route.

Any two hubs can have max of four to 5 routes between them so , we have substantially reduced the number of comparisons. This is why we exploit the startHub:EndHub key.

[Refer to TransitSync_Architecture&Code/transitSyncDriver/app/src/main/java/com/example/transitSyncDriver/SelectRoute.java]

[Note: Add a payment gateway for registration of Routes, a entry fee drivers would have to pay for services provided by the system, every time the driver registers for a new Route they will pay a certain amount, this will also help limit spams and shall also compensate for the computationally expensive process we might have to deploy to keep the system running.]

Another method can be,

GeoHash first and last character of each LatLng and create a String for comparison from Redis
if it exists prompt Route already Exists , Choose Route in Select Route
GeoHash String Example(precision=12)
26.663288251690314, 92.79716634353922->wh9zxtuuftnm
26.66155003156191, 92.79693557552548-> wh9zxtsdv34k
26.661284877011298, 92.79686964180728-> wh9zxts95z9p
GeoHash String = wmwkwp...

A Unique path can have only one specific combination of GeoHash String(So if 90 percentage match means same or similar route don't allow)

One can also deploy their own method for comparison and polyline spam prevention for a route/redundant route insertion considering the possible error or algorithm change in google maps polyline.

For further protection of system, from route spams, we can build a scoring system for each route, First on fresh creation of the route the score will be 0, whenever a new driver enrols to that existing route or a passenger request a driver on that route increment the score by 1. If the score reaches lets say 10 then it means that the route is active and healthy i.e. still in use , we can update the route to intelligent Routing graph database after that, effectively preventing spams and inactive routes.

The route scores can be reset after 3 to 4 months , keeping track of active routes and eliminating closed of routes previously registered, similarly updating the intelligent routing graph databases , which will have two separate graph database , one universal that will contains all routes – active and inactive and one updated every 3 to 4 months, for recommendation.

Cost Analysis and Scalability:

The only cost from Google Api calls, are while route Registration on the driver end of the application, a basic route Registration can be done by just 4 Api calls ->places AutoComplete Api, maps Api, places Api & Directions Api. This can be reduced to 3 Api calls if open Streat maps is Used.

Since, the provider/ System operator and governing body will already charge for Route registration. So, the cost here will be meniscal or might even be profitable.

After route registration the System becomes Isolated, we have effectively stored the route PolyLine in our database(Redis and DynamoDB) , So when a passenger wants to track a route we can provide it ourselves.

So, the main Cost will be incurred in keeping the architecture running for servers, in memory database, Other databases etc.

My personal thoughts:

The ability to track rickshaws, notify drivers of potential passengers, and offer reservation opportunities for drivers other than first in queue will increase the likely hood of earning for drivers , will also reduce congestion , while giving the passengers a benefit of tracking the driver helping them decide to reserve or wait, But the increase in efficiency will be Just on par for the project to be implemented, but the main beauty that would come out of it will be the **Intelligent routing System,** a map of whole network of public transport system, constantly evolving and changing , showing a visual representation of the network highlighting the boon and bane in the system, governments could organize improve and optimize the system for maximum efficiency. Traffic management and deployment or diversion of Vehicles, allotment of more Public transport vehicles on biases of request density, would help govern Traffic. To go to any city in the world and find out the exact Route I need to take to reach my destination would be the most beautiful thing I can think of.

The other major benefit would be with **Buses**, to think of a system where, passengers could keep track of the buses, know how many passengers are inside and book tickets online. Would be marvellous for the public transport System. Saving So much time and increasing efficiency many folds, where a person can just book a cab or reserve from the app itself instead of waiting then seeing a bus completely full and un-boardable.

Feel free to mail at – kryogesh7061@gmail.com