



# PUBLIC TRANSPORT TRACKING APP

K.R MANGLAM UNIVERSITY

COURSE NAME: COMPUTER SCIENCE FUNDAMENTALS AND CAREER  
PATHWAYS

COURSE CODE: ETCCCP105

SUBMITTED TO: MR. RAJESH KUMAR

SUBMITTED BY: MANAS BHASKER

PROGRAMME: B.TECH CSE CORE

ROLL NO: 2501010226

## INTRODUCTION:

Currently, public transports have become a necessity for many people like those who do not own a personal vehicle or for those who cannot afford one, in that aspect public transport make a huge difference in these scenarios, even if you do own a vehicle public transport helps reducing the overall pollution emission and the traffics as there will be few vehicles on roads. And there are many problems like noise pollution, fewer accidents etc. Which public transport can manage to tackle if used correctly.

This project aims to fix one of the core issues of public transport, that is, it not being feasible for many people. Suppose you are on your way to your destination, and you are going to use a bus and walk towards the bus stop, there you wait for half an hour not knowing the exact time of arrival of the bus. And when the bus arrives there is no seat for you.

This is where my app (or so it can be) comes in using the computational thinking I have made an algorithm which allows users to tackle these problems.

Using abstraction, decomposition and pattern recognition, this project aims to fix the things that are problematic in the upper given scenario.

It allows users to:

1. Check the current location of the vehicle.
2. Check the ETA (estimated time of arrival) of the vehicle.
3. And the stops that it would take in the predetermined route.

By solving these problems, the project provides solutions to users for properly managing thier rides and saving a whole lot of their time.

Analysis of the problem statement is done by using the abstraction, decomposition and pattern solving in the following.

## ABSTRACTION:

**Abstraction** involves reducing complexity by focusing on the main idea and ignoring specific details. For this app, the core abstraction is the "**real-time tracking of a public transport vehicle.**"

- **Vehicle:** A bus, train, or car is an entity with a specific location, a route, a schedule, and a unique identity.
- **User:** An individual with a device (e.g., a smartphone) who wants to know the location of a vehicle relative to their own location or a specific location.
- **Route:** A predetermined sequence of stops that a vehicle follows. Each route has a starting point, a destination, and a set of intermediate stops (e.g., washroom breaks or somewhere the user has requested a stop).
- **Location:** A set of coordinates (latitude and longitude) that can be used to pinpoint the position of both the vehicle and the user.
- **Real-time data:** Information that is constantly updated and reflects the current state of the system, such as a vehicle's current location and its estimated time of arrival (ETA) at upcoming stops. (most crucial part of this application)

By abstracting these concepts, we can create a simplified model that focuses on the relationships between these elements: a user wants to view a vehicle's real-time location on a specific route, and they are interested in its ETA at a particular stop.

## DECOMPOSITION:

**Decomposition** is the process of breaking down a large, complex problem into smaller, more manageable sub-problems. The public transport tracking app can be broken down into the following key components:

1. **Frontend (User Interface):** This is what the user interacts with.
  - a. **Map view:** A map displaying the user's location, vehicle locations, and routes.
  - b. **Search functionality:** Allows users to search for routes, stops, or specific vehicles.
  - c. **Route and vehicle details:** Displays information like the route name, vehicle number, and a list of upcoming stops.
  - d. **User feedback:** Features for reporting issues or providing feedback on the data accuracy.

2. **Backend (Server-side logic):** This is the core of the application that handles data processing and management.
  - a. **Data Ingestion:** A system for receiving real-time location data from public transport vehicles. This data could come from GPS devices, APIs from transport authorities, or other sources.
  - b. **Data Processing:** Logic to process the raw GPS data, calculate ETAs, and match vehicles to their respective routes and stops.
  - c. **Database:** Stores static data (routes, stops, schedules) and dynamic data (vehicle locations, traffic conditions).
  - d. **API (Application Programming Interface):** A set of rules and protocols that allows the frontend to request and receive data from the backend. This is how the app gets the real-time location of the buses.
3. **Data Sources:** The origin of real-time data.
  - a. **GPS Devices:** Hardware installed in public transport vehicles that transmit their location.
  - b. **Transport Authority APIs:** Many public transport agencies provide real-time data feeds. The app would need to connect to and parse this data.
  - c. **Crowdsourcing:** A less reliable but possible method where users can manually report vehicle locations.

## **PATTERN RECOGNITION:**

**Pattern recognition** involves identifying recurring similarities and regularities in the problem. Recognizing these patterns helps in designing reusable and efficient solutions.

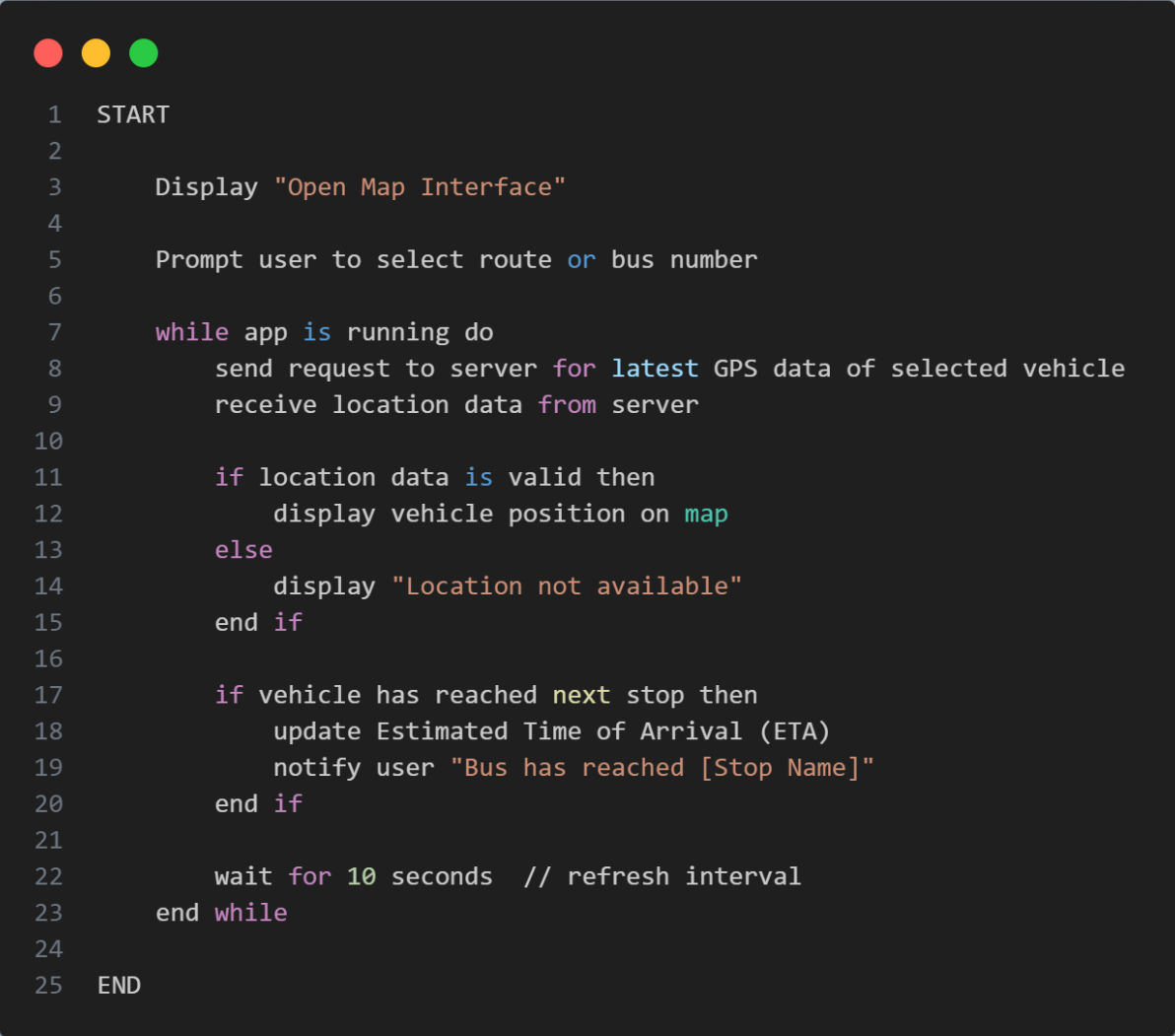
1. **Real-time Location Updates:** The core pattern is the continuous updating and display of location data. This is a common pattern in many modern apps, such as ride-sharing services (Uber, Lyft) and delivery tracking apps. The process involves:
  - a. A device (the bus's GPS) sending its location to a server at regular intervals.
  - b. The server is processing this data.
  - c. The user's app makes repeated requests to the server for the latest location data. This is a classic **client-server model**.
2. **Map-based Visualization:** The visualization of moving objects on a map is a standard pattern in location-based services. This requires using a mapping service API (e.g., Google Maps, Mapbox) and overlaying custom markers for vehicles and routes. The **marker movement** on the map is a key visual pattern.

3. **Route Following:** A vehicle's location can be plotted against a predefined route path. The pattern here is to track the vehicle's progress along a linear path (the route) and predict its arrival time at the next stop based on its current speed and the distance remaining. This involves using **geospatial algorithms** to calculate distances and ETAs.
4. **Data Structure:** The data for each entity (vehicle, route, stop) follows a consistent structure.
  - a. **Vehicle:** ID, location (lat, lon), routeID, speed.
  - b. **Route:** ID, name, list of stops.
  - c. **Stop:** ID, name, location (lat, lon). Recognizing this pattern allows for the creation of a **relational database schema** to store and manage the information efficiently.

By applying these three analytical techniques, the seemingly complex problem of building a real-time public transport tracking app is broken down into manageable, understandable, and solvable components, paving the way for a structured development process.

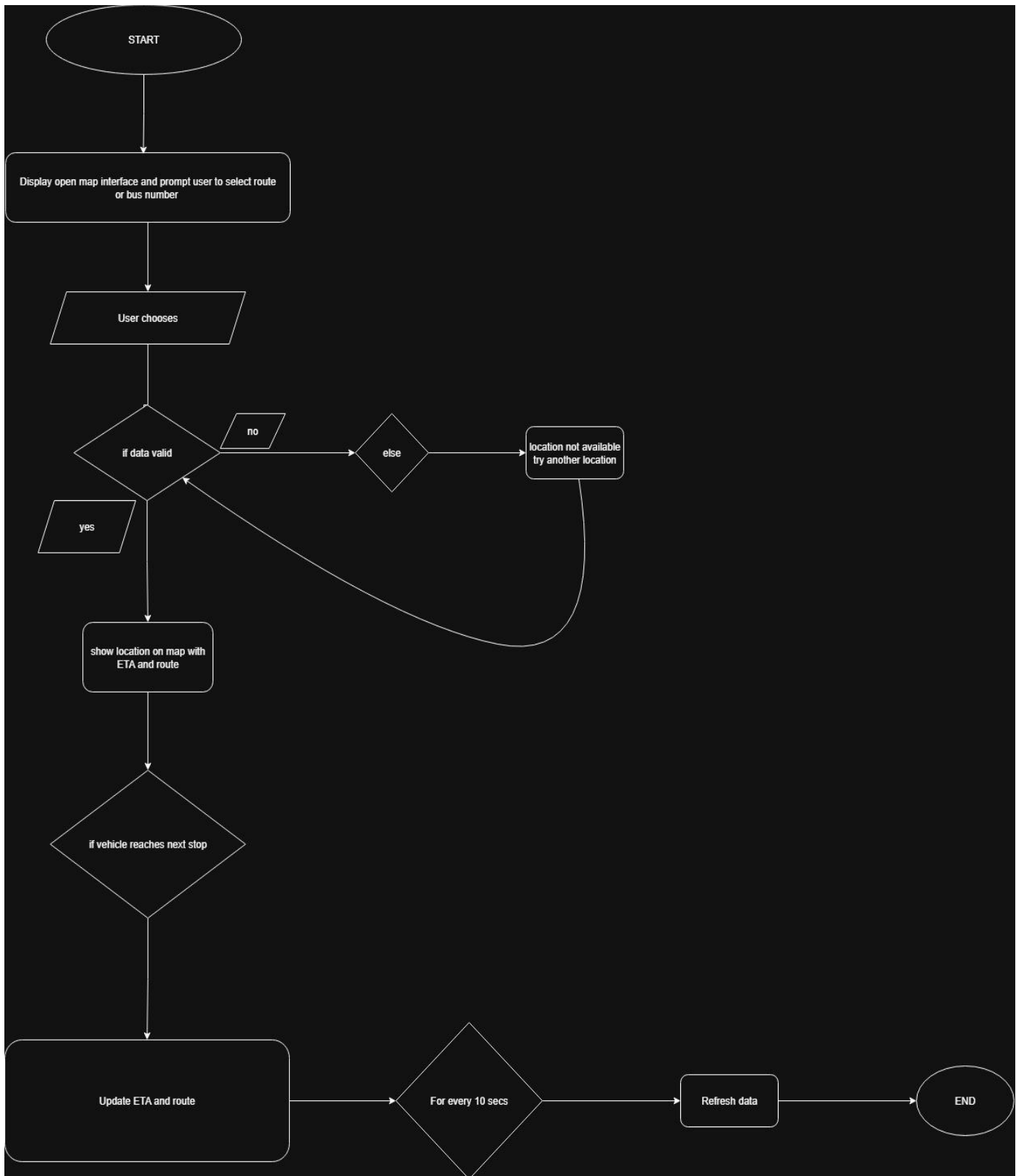
SOLUTION DESIGN:

1. Pseudocode – representation of process using proper conventions (e.g., indentation, uppercase for keywords, meaningful variable names).



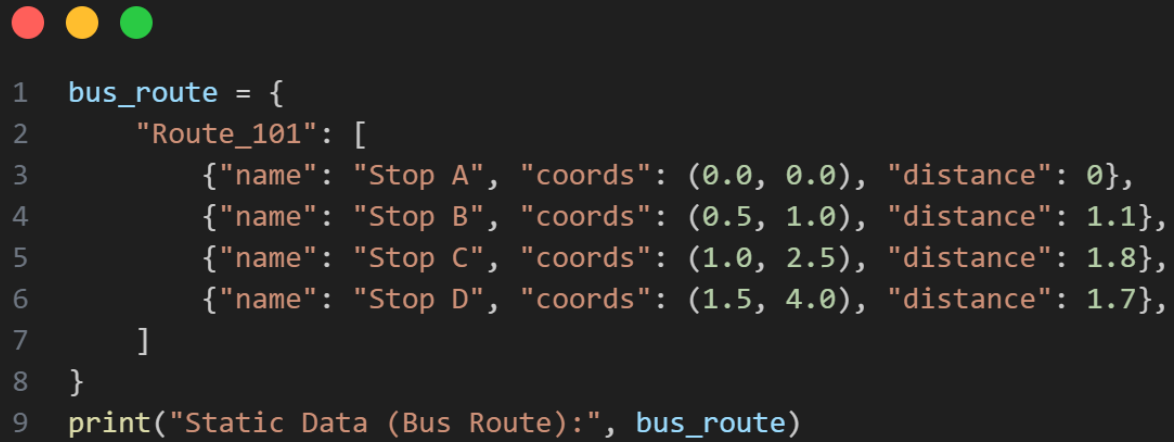
```
1  START
2
3      Display "Open Map Interface"
4
5      Prompt user to select route or bus number
6
7      while app is running do
8          send request to server for latest GPS data of selected vehicle
9          receive location data from server
10
11         if location data is valid then
12             display vehicle position on map
13         else
14             display "Location not available"
15         end if
16
17         if vehicle has reached next stop then
18             update Estimated Time of Arrival (ETA)
19             notify user "Bus has reached [Stop Name]"
20         end if
21
22         wait for 10 seconds // refresh interval
23     end while
24
25  END
```

2. Flowchart: using standard symbols to depict the complete process logically.



## IMPLEMENTATION:

The following snippet will be of the distance calculated of vehicle coded in python:

A screenshot of a code editor window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code is a Python dictionary representing a bus route. It starts with a line number '1' followed by 'bus\_route = {'. Line '2' has an indented 'Route\_101': ['. Line '3' has an indented dictionary entry for 'Stop A' with coordinates (0.0, 0.0) and distance 0. Line '4' has an indented dictionary entry for 'Stop B' with coordinates (0.5, 1.0) and distance 1.1. Line '5' has an indented dictionary entry for 'Stop C' with coordinates (1.0, 2.5) and distance 1.8. Line '6' has an indented dictionary entry for 'Stop D' with coordinates (1.5, 4.0) and distance 1.7. Line '7' has an indented closing bracket ']' for the list. Line '8' has a closing brace '}' for the dictionary. Line '9' has 'print("Static Data (Bus Route):", bus\_route)'.

```
1 bus_route = {
2     "Route_101": [
3         {"name": "Stop A", "coords": (0.0, 0.0), "distance": 0},
4         {"name": "Stop B", "coords": (0.5, 1.0), "distance": 1.1},
5         {"name": "Stop C", "coords": (1.0, 2.5), "distance": 1.8},
6         {"name": "Stop D", "coords": (1.5, 4.0), "distance": 1.7},
7     ]
8 }
9 print("Static Data (Bus Route):", bus_route)
```



## OUTPUT:

```
PS C:\Users\user\Desktop\assignmentCSPW> & 'c:\Users\user\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\user\.vscode\extensions\ms-python.debu
gpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '53040' '--' 'c:\Users\user\Desktop\assignmentCSPW\code.py'
Static Data (Bus Route): {'Route_101': [{'name': 'Stop A', 'coords': (0.0, 0.0), 'distance': 0}, {'name': 'Stop B', 'coords': (0.5, 1.0), 'distance': 1.1}, {'n
ame': 'Stop C', 'coords': (1.0, 2.5), 'distance': 1.8}, {'name': 'Stop D', 'coords': (1.5, 4.0), 'distance': 1.7}]]}
PS C:\Users\user\Desktop\assignmentCSPW> []
```

## REFLECTION:

While working on this project I was stuck on many problems like:

- How do I write text beneath an image.
- How do I improve the overall quality of this report.
- How do I take the snippet of the code.
- How do I make my pseudocodes and codes efficient and better.

To solve these hurdles, I resorted to using a simple and quick web search which gave me most of my answers,

When it came to making my codes efficient and better i used ChatGPT AI to better my prewritten code and pseudocode which gave me a better understanding of overall code and helped me understand various new things, like how https plays a major role conversing between user and the backend servers and various things that I was ignorant of.

I am considering of changing many things in this project and introducing new ones like:

- Offline mode
- Text to speech for visually challenged

And many more.

To conclude, I had fun making this project and racking my brains till i eventually got the results that I desired, with my current knowledge I have limitations. But i hope to make this project reach its full potential.