Design Document

1 System Overview

The system comprises of 3 components –

1. Backend Server
2. Frontend Mobile Client
3. Jetson SLAM Device

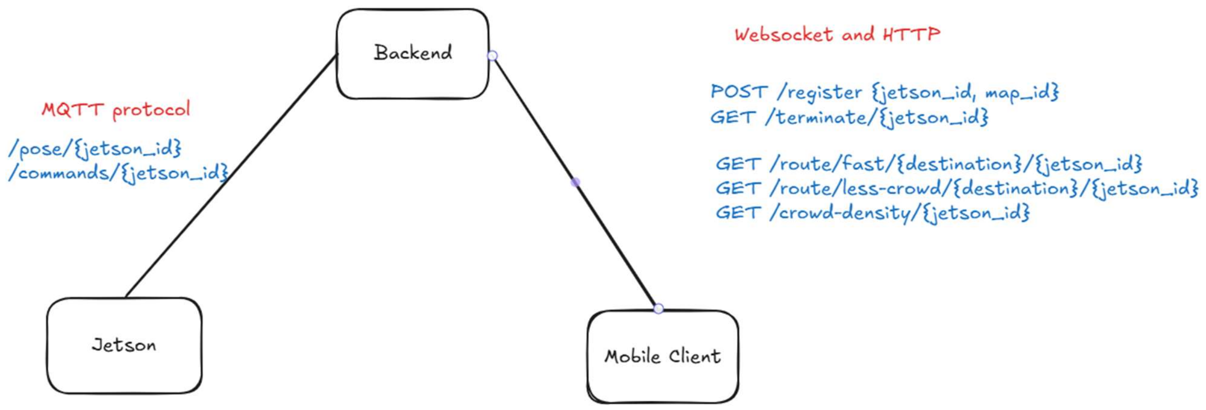
2 Communication Flow

2.1 Jetson – Backend

- Communicate via the **MQTT** protocol.
- I am currently using a public MQTT broker; I can migrate it to a self-hosted private broker if required.
- Backend publishes commands to the jetson via **/commands /{jetson_id}**.
- Jetson publishes pose to the backend via **/pose/{jetson_id}**.
- The backend subscribes to these topics, processes incoming data and stores user location information in a Redis database for rapid retrieval.

2.2 Backend – Mobile Client

- Real time communication using **WebSockets.**
- The backend pushes live pose updates and route information directly to the corresponding mobile clients.

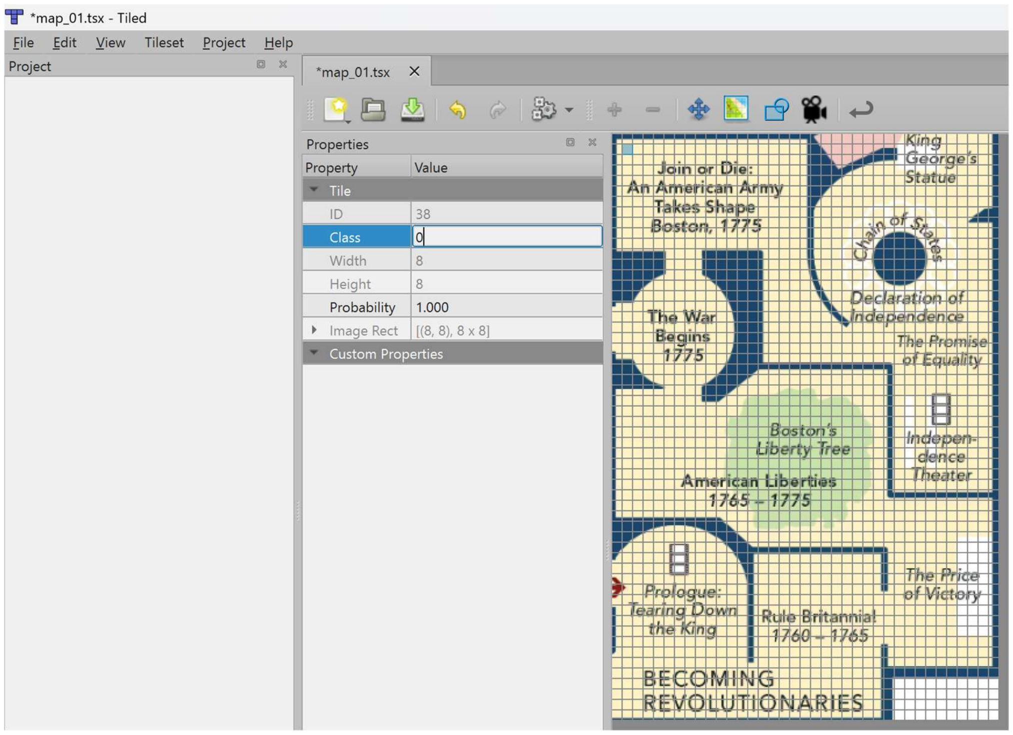


Backend
MQTT protocol
/pose/{jetson_id}
/commands/{jetson_id}
Websocket and HTTP
POST /register {jetson_id, map_id}
GET /terminate/{jetson_id}
GET /route/fast/{destination}/{jetson_id}
GET /route/less-crowd/{destination}/{jetson_id}
GET /crowd-density/{jetson_id}
Jetson
Mobile Client

3 Core Functionalities and Considerations

3.1 Grid Based Map Representation

- The map is divided into a grid 8x8 pixel tiles to optimize computational efficiency. This was done to simplify the pathfinding by reducing the search space and makes ID identification to individual tiles easier and less time consuming.
- **Tiled** software was used to assign unique IDs to each grid tile. (1 for obstacle, 0 for free, and other numbers for different landmarks in that map)



Property	Value
Tile	
ID	38
Class	0
Width	8
Height	8
Probability	1.000
Image Rect	[(8, 8), 8 x 8]
Custom Properties	

3.2 Crowd Density Calculations (Heatmap)

- The latest user location data is fetched from the redis database.
- Crowd density is calculated per grid cell across the map.
- This data is transmitted to the frontend and rendered as heatmap blobs, darker blobs indicate higher crowd density.

3.3 Path Planning

- User can choose between 2 options – either the fastest route or the route with least obstacles.
- Path planning is performed on the backend using A* algorithm.
- Crowd density data is integrated into the A* cost function to penalise routes through crowded areas.

3.4 Route Deviation Detection

- This check is implemented on the frontend (mobile) itself.
- Basic distance threshold checks detect when a user deviates from the suggested route.
- Upon detecting deviation, a Reroute button is displayed on the user for recalculating the path.

3.5 Other Considerations

- I coded in such a way that the backend can handle multiple users and multiple maps concurrently.
- However, all the map-related data (image, pixel dimensions, landmarks, landmark co-ordinates etc) is hardcoded in the backend and frontend codebase.
- I was thinking about moving this info to a database, allowing them to query this database for necessary info.

4 Tech Stack Used

1. Jetson SLAM client – python
2. Backend Server – python (FastAPI)
3. Mobile client – react native