

Assignment: Building "Movi" - The Multimodal Transport Agent

Project Scenario

Welcome to MoveInSync! We are building a next-generation, supply-first admin console called **MoveInSync Shuttle**. Our platform operates on a clear, layered logic:

- **Layer 1 (Static Assets):** Transport managers first define the "static" assets.
 1. **Stops** are created (e.g., "Gavipuram", "Peenya").
 2. A **Path** is created as an ordered sequence of **Stops**
 3. A **Route** is created by assigning a specific time to a **Path**. This is all managed in our **manageRoute** page.
- **Layer 2 (Dynamic Operations):** On a daily basis, managers perform "dynamic" operations on the **busDashboard**. They **Deploy** vehicles onto these **Routes**, which then generates **Trips**, and finally, daily **Trip-Sheets** for drivers and riders.

Your task is to build a working prototype of our new AI assistant, **"Movi"**.

Movi is a "knowledge-aware" AI agent that helps transport managers perform their complex tasks. It's a true multimodal assistant that understands **voice, text, images, and video** to process commands and learn from our product documentation.

Critically, Movi understands this **Stop -> Path -> Route -> Trip** data flow and the **consequences** of changing dynamic operations (e.g., removing a vehicle from a **Trip** will affect bookings and break the **Trip-Sheet** generation).

Core Agent Architecture Requirement

The backend for the Movi assistant **must** be built using **langgraph**. We are specifically testing your ability to build a robust, stateful, and multi-step agent that can manage conditional logic (like checking for consequences), orchestrate multiple tools, and handle complex user interactions.

Core Objective

Deliver a working web application prototype that includes:

1. A dummy database of our transport operations based on the **Stop -> Path -> Route** logic.

2. A basic, functional **two-page admin UI** (cloned from our screenshots) that displays and manages data from the database.
 3. A **multimodal AI assistant ("Movi")** integrated into the UI, powered by a **langgraph** agent.
-

Part 1: The Data Layer (Dummy Database)

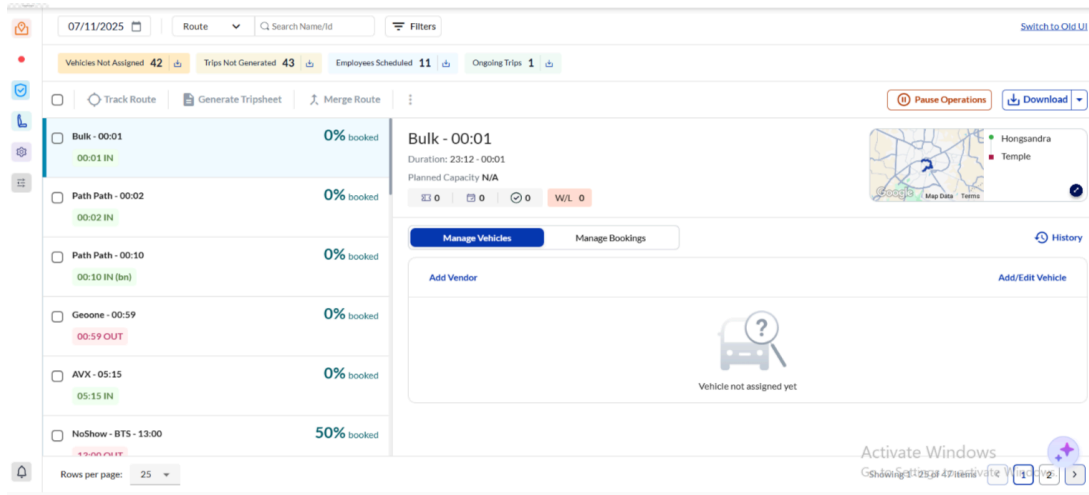
Your first task is to design and populate a simple database that reflects our precise operational flow.

- **Technology:** You are free to use any database (e.g., PostgreSQL, MySQL, SQLite, MongoDB).
 - **Task:**
 1. **Define a schema** that can store the following interconnected data:
 - **Static Assets (for **manageRoute** page):**
 - **Stops** (e.g., **stop_id**, **name**, **latitude**, **longitude**)
 - **Paths** (e.g., **path_id**, **path_name**, **ordered_list_of_stop_ids**)
 - **Routes** (This table combines a Path + Time, as seen in the UI): **route_id**, **path_id**, **route_display_name** (e.g., "Path2 - 19:45"), **shift_time**, **direction**, **start_point**, **end_point**, **status** ('active', 'deactivated')
 - **Dynamic Assets & Operations (for **busDashboard** page):**
 - **Vehicles** (e.g., **vehicle_id**, **license_plate**, **type** (Bus/Cab), **capacity**)
 - **Drivers** (e.g., **driver_id**, **name**, **phone_number**)
 - **DailyTrips** (The items in the left panel like 'Bulk - 00:01'): **trip_id**, **route_id**, **display_name**, **booking_status_percentage**, **live_status** (e.g., '00:01 IN')
 - **Deployments** (The link between vehicles and trips): **deployment_id**, **trip_id**, **vehicle_id**, **driver_id**
 2. **Populate** your database with realistic dummy data to fill both pages as seen in the screenshots. Ensure your **Routes** table data is derived from **Paths** and **Stops**.
-

Part 2: The Context Layer (Admin Console UI)

You need to build a simple UI for the transport manager similar to the provided screenshots. Movi must be present and context-aware on both pages.

- **Technology:** You are free to use any web framework (e.g., React, Vu



Angular).

Note: You are **allowed and encouraged** to use AI coding assistants (e.g., GitHub Copilot, Cursor, etc) for the UI and the DB part. We are evaluating your AI agent architecture, not your HTML/CSS skills.

- **Task:**

1. **Page 1: busDashboard**

2. **Page 2: manageRoute**

Route ID	Route Name	Direction	Shift Time	Route Start Point	Route End Point	Capacity	Allowed Waitlist	Action
76918	Path2 - 19:45	LOGIN	19:45	Gavipuram	peenya	0	0	
76919	Path2 - 23:00	LOGIN	23:00	Gavipuram	peenya	0	0	
76917	Path2 - 20:00	LOGIN	20:00	Gavipuram	peenya	0	0	
76920	Path2 - 19:00	LOGIN	19:00 (KS)	Gavipuram	peenya	0	0	
76914	Path1 - 21:00	LOGIN	21:00	Gavipuram	Temple	0	0	
76913	Path1 - 20:00	LOGIN	20:00	Gavipuram	Temple	0	0	
76916	Path1 - 23:00	LOGIN	23:00	Gavipuram	Temple	0	0	
76915	Path1 - 22:00	LOGIN	22:00	Gavipuram	Temple	0	0	
76234	paradise - 05:00	LOGIN	05:00	BTM	NoShow	6	0	
76912	Dice	LOGIN	18:00	BTM	NoShow	6	0	

3. **UI-Context-Aware AI:** Movi's chat interface must be present on **both** pages. The agent's **langgraph** state should be aware of which page the user is currently on (e.g., by you passing **currentPage**: 'busDashboard').
 - **Example:** If the user is on the **manageRoute** page and asks, "Help me create a new route," Movi should provide guidance relevant to that page.
-

Part 3: The Intelligence Layer ("Movi" **langgraph** Agent)

This is the core of the assignment. You will build the Movi AI assistant.

Movi's Required Capabilities (to be orchestrated by **langgraph):**

1. **Voice & Text I/O:**
 - **Input:** The user must be able to interact with Movi using **both text and voice** (Speech-to-Text).
 - **Output:** Movi's responses should be rendered as **text in the chat and spoken aloud** (Text-to-Speech).
2. **Perform >10 Actions:** Movi must be able to understand user intent and perform *at least 10 distinct actions* (as tools/functions called by your graph) that reflect the new data model.
 - **Examples (You must implement these):**
 - **Read (Dynamic):** "How many vehicles are not assigned?"
 - **Read (Dynamic):** "What's the status of the 'Bulk - 00:01' trip?"
 - **Read (Static):** "List all stops for 'Path-2'."
 - **Read (Static):** "Show me all routes that use 'Path-1'."
 - **Create (Dynamic):** "Assign vehicle 'MH-12-3456' and driver 'Amit' to the 'Path Path - 00:02' trip."
 - **Delete (Dynamic):** "Remove the vehicle from 'Bulk - 00:01'."
 - **Create (Static):** "Create a new stop called 'Odeon Circle'."
 - **Create (Static):** "Create a new path called 'Tech-Loop' using stops [Gavipuram, Temple, Peenya]."
 - ...you must add at least 2 more actions of similar complexity.
3. 🧠 **"Tribal Knowledge" (The **langgraph** "Consequence" Flow):** This is the most critical requirement for your **langgraph** implementation. Your agent must model our product logic.
 - **Example Scenario:**
 - **User:** (On the **busDashboard**) "Movi, remove the vehicle from 'Bulk - 00:01'."
 - **langgraph State:** Your graph should move from [**agent_entry**] to a [**check_consequences**] node.

- **[check_consequences] Node:** This node queries the DB and finds that this `trip` is `25% booked` (or has any bookings) and is scheduled to run. It updates the `agent_state` with this critical info.
 - **langgraph Edge:** The graph must conditionally route to a `[get_confirmation]` node instead of an `[execute_action]` node.
 - **Movi Response (from [get_confirmation]):** "I can remove the vehicle. However, please be aware the 'Bulk - 00:01' trip is already 25% booked by employees. Removing the vehicle will **cancel these bookings** and a trip-sheet will fail to generate. **Do you want to proceed?**"
 - **langgraph State:** The graph now *persists* in this state, waiting for the user's "yes/no" response.
4. **True Multimodal Input (Mandatory):** Your `langgraph` agent must orchestrate tools that process these inputs.
- **Image Input:**
 - **Example:** The user uploads a screenshot of the `busDashboard` and says, "Remove the vehicle from this trip" (the screenshot could have a red arrow pointing to the "Bulk - 00:01" row). Your agent must use a vision tool to identify the trip, update its state, and trigger the "Tribal Knowledge" flow.
-

Deliverables

1. A **Git repository** (GitHub, GitLab) containing your full source code.
2. A detailed **README.md** file that includes:
 - **Architecture & langgraph Graph Explanation:**
 - A brief description of your overall architecture (frontend, backend, DB).
 - **Crucially:** A detailed explanation of your **langgraph design**. You must describe your `agent_state`, your main `nodes`, and the `conditional_edges` you used to manage the "Tribal Knowledge" and multimodal flows. A simple diagram is highly recommended.
 - **Setup Instructions:** Instructions on how to install dependencies and run the project locally.
3. A **short demo video** (2-5 minutes) showing your complete, working application. You **must** demonstrate:
 - Both UI pages (`busDashboard` and `manageRoute`) functionally working.
 - The "Tribal Knowledge" `langgraph` flow (showing the "consequence" question).
 - The **Image Input** feature (e.g., uploading a screenshot of the `busDashboard`).
 - The **UI-Context-Awareness** (asking for help on the `manageRoute` page).

Good luck! We are excited to see your `langgraph` agent in action.