Engineering Take Home Interview:

🛺 Ride Dispatch System – Technical Assessment

Design and implement a simplified ride-hailing backend system using **FastAPI**, along with a **basic frontend** to visualize and simulate the system. The platform should operate in a grid-based city where riders request rides, and drivers are dispatched based on ETA, fairness, and availability.

You'll build the system from scratch — backend, logic, and UI

🎯 Objectives

You will implement:

- A FastAPI backend to manage the city grid, drivers, riders, and ride requests
- Dispatch logic to assign drivers to ride requests
- A fallback mechanism when drivers reject rides
- A **simple frontend UI** to:
  - Visualize the grid
  - Add/remove drivers and riders
  - Request rides
  - Advance simulated time (`tick`)
  - See current system state

🗺️ Environment

- The city is a 2D grid (e.g., 100×100).
- Drivers and riders exist at `(x, y)` coordinates.

- Time advances manually through a `/tick` endpoint.
- Movement happens one unit per tick unless you document otherwise.
- No real-time behavior or persistence is needed — in-memory simulation is fine.

🧩 System Requirements

1. Entities

Driver

- Unique ID
- Current location `(x, y)`
- Status: `available`, `on_trip`, or `offline`

Rider

- Unique ID
- Pickup location `(x, y)`
- Dropoff location `(x, y)`

RideRequest

- Rider ID
- Pickup and dropoff locations
- Status: `waiting`, `assigned`, `rejected`, `completed`, or `failed`

---

2. Flow

- A rider requests a ride via the API or frontend.
- The system finds the **best available driver** based on the dispatch logic goals defined below.
- The driver can accept or reject.

- If rejected, the system tries the next-best driver.
- Once accepted, the driver moves toward pickup → dropoff location.
- Movement updates occur on each `/tick`.

3. Dispatch Logic Goals

Design your dispatch logic to balance:

Goal	Description
Low ETA	Assign drivers with shortest travel time to pickup
Fairness	Ensure no driver gets all requests; distribute evenly
Efficiency	Maximize fulfilled rides; minimize idle drivers
Fallbacks	Retry with other drivers if the first one rejects

You may define and document your own algorithms — creativity is encouraged!

💻 Frontend Requirements

You must build a **basic browser-based UI** (no need for styling). It should let users:

- Add/remove drivers and riders
- Request a ride
- Trigger the next time tick (e.g., a "Next Tick" button)
- Visualize drivers, riders, and trips on the grid

Any front end framework is fine.

📦 Deliverables

Submit a GitHub repo or zip file containing:

- ✅ FastAPI backend
- ✅ Basic frontend UI
- ✅ README with:
    - How to run the system

    - How your dispatching works
    - Any assumptions or simplifications you made

🧪 Evaluation Criteria

Category	What We're Looking For
✅ Correctness	Are ride requests assigned and completed correctly?
🧠 Dispatch Logic	Is your logic well-thought-out and documented?
🧹 Code Quality	Is the code clean, modular, and understandable?
🔁 Extensibility	Is your system designed in a way that could scale or support future features?

🏷️ Notes

- You **do not** need authentication, persistent databases, or real-time sockets.
- The system can run entirely locally and use in-memory storage.
- You may define default driver speed, ETA calculation method, or rejection behavior — just explain all assumptions in the README.

Good luck! We're excited to see your approach and engineering decisions 🚀