

RapidRide — Task Allocation (Refined Version)

■ PROJECT CONTEXT (Copy-Paste Block for Any LLM)

This project is a distributed ride-hailing simulation platform named RapidRide.

It includes multiple coordinated subsystems that must integrate cleanly across all team members.

All teammates must strictly follow the module boundaries and API expectations listed below.

- Backend: Node.js (core ride flow + WS gateway)
- Microservices: FastAPI (ML, fare, ETA)
- Real-time infra: Redis Pub/Sub + WebSockets
- Async tasks: RabbitMQ
- Search + logs: Elasticsearch
- Observability: Prometheus + Grafana + ELK

Necessary Global Rules

- Use JSON for all inter-service communication
- Naming conventions: snake_case for Python, camelCase for JS
- All services expose health check endpoints
- Use environment variables for credentials
- Follow the shared API contract

■ Shared Integration Contract

All teammates must follow this exact contract to prevent integration issues:

- FastAPI returns { fare, eta, distance_km }
- Redis stores driver state under key: driver:{id}
- WebSocket events use channels: rider:{id}, driver:{id}
- Trip logs stored in ES index: rapidride_trips

Input/Output Consistency Checklist

- Timestamps in ISO format
- Distance in km (float)
- Fare as float

- All IDs as strings

■ Detailed Task Allocation (Pasteable Block for Each Teammate)

Each teammate gets a structured work package with prerequisites, tasks, and expected outputs.

The sections below are written in a way they can be pasted into any LLM to start coding immediately.

■ Sampath — Real-Time Backend (WebSockets + Redis)

Tasks:

- Implement WebSocket gateway handling rider/driver events
- Store and update driver states in Redis
- Handle ride_start, ride_end, location_update events
- Integrate Node backend with RabbitMQ for async jobs

Deliverables:

- ws.js WebSocket server
- redis.js driver caching utilities
- eventSchemas.md documenting WS events

Integration Requirements

- WS messages must match global schema
- Driver state format consistent across modules
- Redis channels follow naming: driver_location_updates

■ Anurag — FastAPI + Machine Learning + RabbitMQ

Tasks:

- Build FastAPI routes /fare, /eta, /reverse-geocode
- Train ML model for ETA prediction
- Implement RabbitMQ consumers for heavy tasks
- Return consistent structured responses

Deliverables:

- ml_model.pkl, preprocess.py, inference_service.py
- fastapi_app.py
- rabbit_workers.py

Integration Requirements

- Return values must follow shared schema
- FastAPI must expose health-check route /health
- Model inputs: distance_km, traffic_level, historical_mean_eta

■ Adarsh — Observability & Logging (Prometheus + Grafana + ELK)

Tasks:

- Create Prometheus exporters for backend and FastAPI
- Build Grafana dashboards for ride flow & latency
- Set up ELK pipeline for logs
- Index trip logs + WS logs into ES

Deliverables:

- prometheus.yml + exporters
- grafana_dashboard.json
- logstash.conf + index templates

Integration Requirements

- All logs in JSON
- Trip logs follow ES index schema
- Dashboards must read Prometheus metrics

■ Sanjay — Frontend (React + WebSocket Client + Maps)

Tasks:

- React UI for rider & driver interfaces
- WebSocket client to listen for events
- Map view with driver movement
- Ride request form integrated with backend

Deliverables:

- React components
- eventHandlers.js
- MapView.jsx

Integration Requirements

- Must consume WS messages exactly as defined
- Frontend environment variables must map to backend URLs

- Inputs validated before API calls

■ Avinash — Scraping + ElasticSearch + Trip Search

Tasks:

- Scrape fare and competitor data
- Clean + normalize scraped datasets
- Index trip logs into ElasticSearch
- Expose /search-trip API

Deliverables:

- scraper.py
- clean_dataset.py
- es_indexer.py + es_schema.json

Integration Requirements

- ES index name: rapidride_trips
- Distance + fare units consistent across services
- Scraped data stored in /data/raw