AI assistant coding lab test 03

Hall ticket no:2403a52049

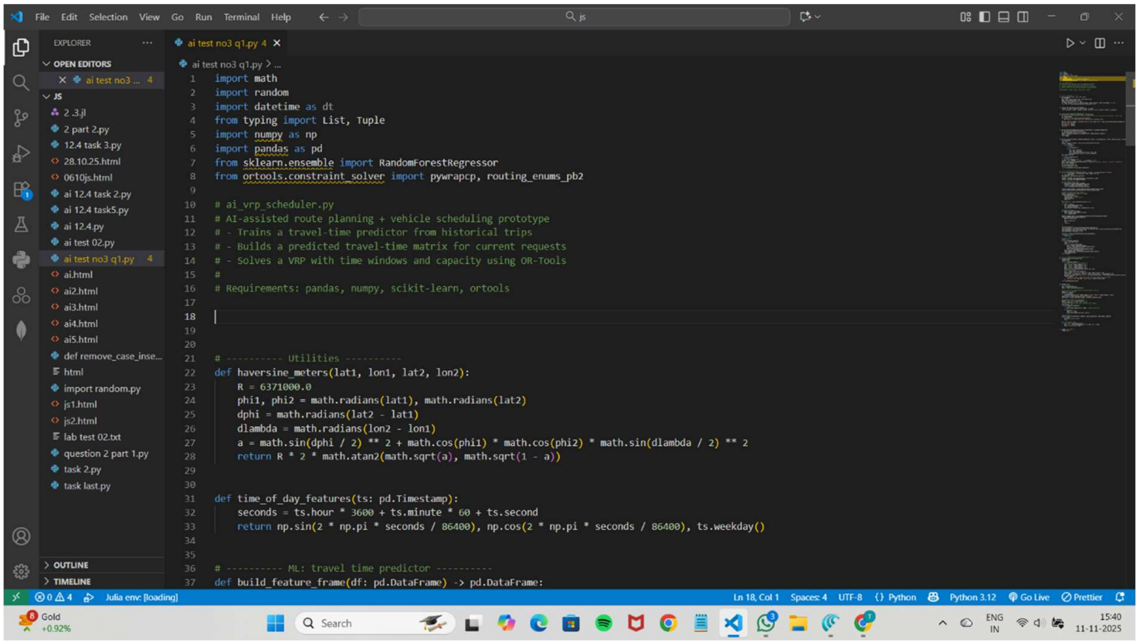
Batch no:03

Name: swaranjith reddy thatipalli

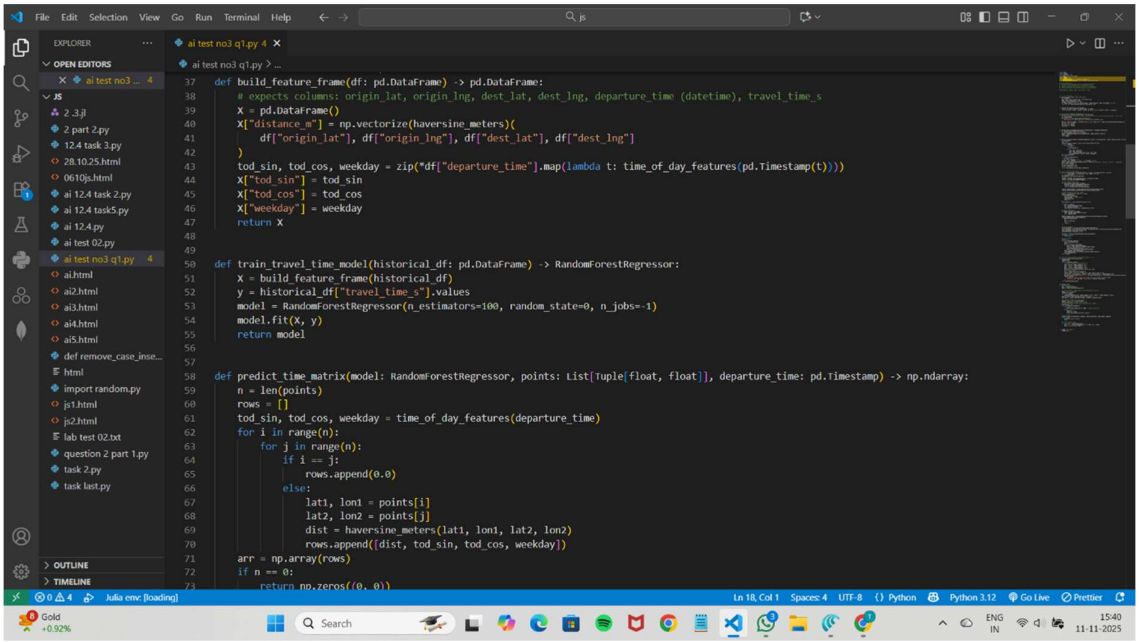
TASK _01

Prompt:

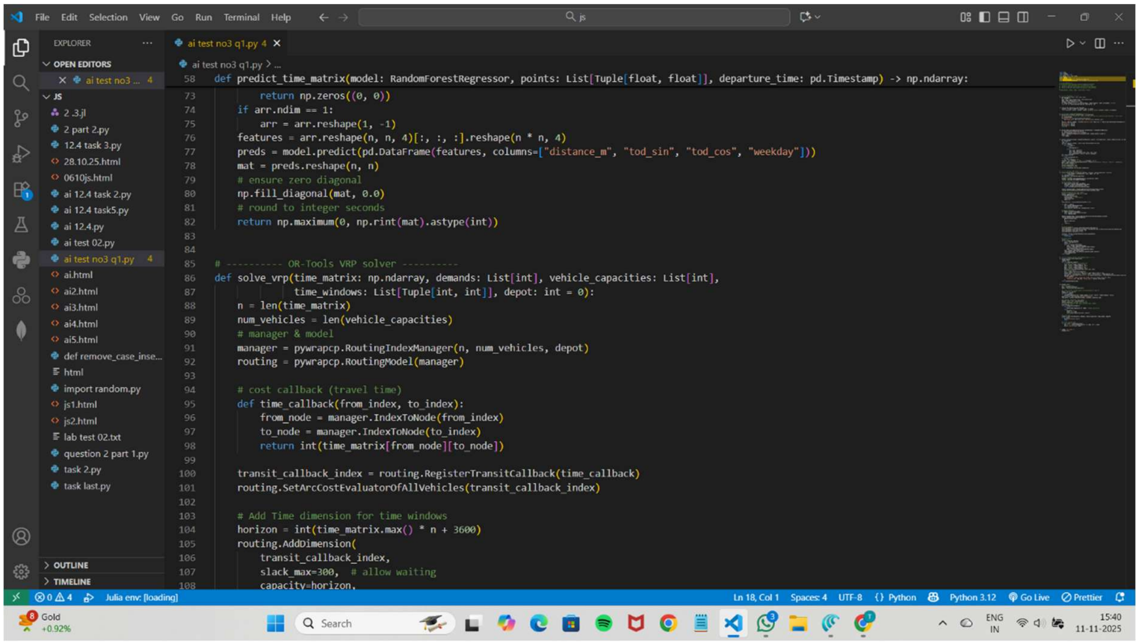
You are an expert AI engineer working in the domain of Transportation.
A transportation company is facing a challenge related to algorithm optimization with AI assistance specifically, improving route planning and vehicle scheduling efficiency.


File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
ai test no3 q1.py 4
ai test no3 q1.py > ...
Ln 18, Col 1 Spaces: 4 UTF-8 { } Python Python 3.12 Go Live Prettier
Julia env: [loading]
Search
15:40
11-11-2025

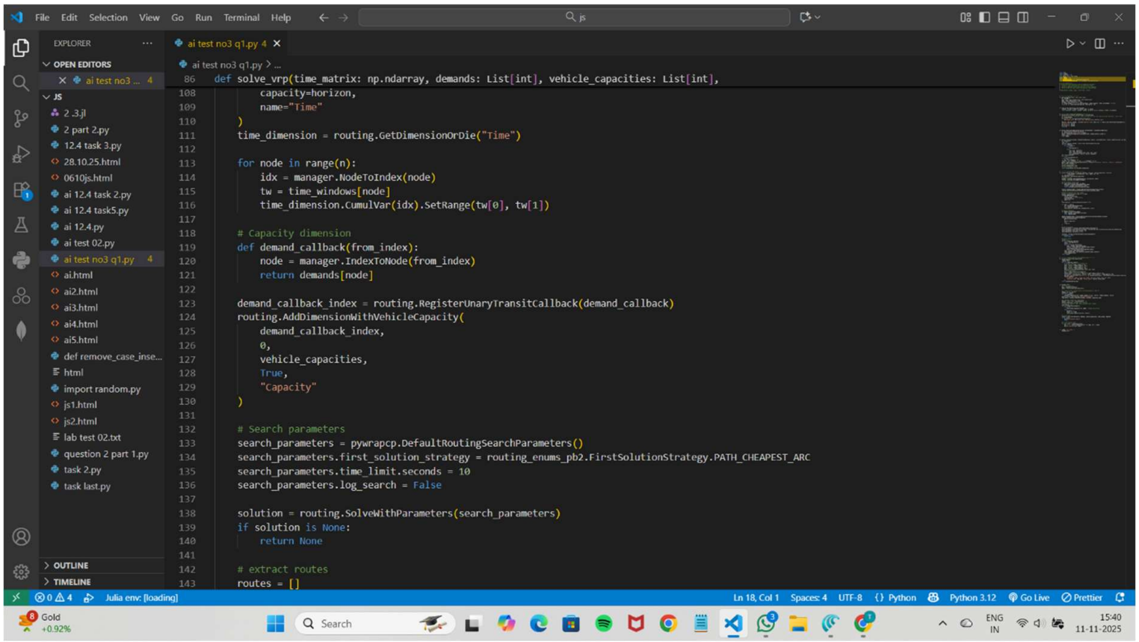
```python
import math
import random
import datetime as dt
from typing import List, Tuple
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from ortools.constraint_solver import pywrapcp, routing_enums_pb2

# ai_vrp_scheduler.py
# AI-assisted route planning + vehicle scheduling prototype
# - Trains a travel-time predictor from historical trips
# - Builds a predicted travel-time matrix for current requests
# - Solves a VRP with time windows and capacity using OR-Tools
#
# Requirements: pandas, numpy, scikit-learn, ortools




# ---------- Utilities ----------
def haversine_meters(lat1, lon1, lat2, lon2):
    R = 6371000.0
    phi1, phi2 = math.radians(lat1), math.radians(lat2)
    dphi = math.radians(lat2 - lat1)
    dlambda = math.radians(lon2 - lon1)
    a = math.sin(dphi / 2) ** 2 + math.cos(phi1) * math.cos(phi2) * math.sin(dlambda / 2) ** 2
    return R * 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))


def time_of_day_features(ts: pd.Timestamp):
    seconds = ts.hour * 3600 + ts.minute * 60 + ts.second
    return np.sin(2 * np.pi * seconds / 86400), np.cos(2 * np.pi * seconds / 86400), ts.weekday()


# ---------- ML: travel time predictor ----------
def build_feature_frame(df: pd.DataFrame) -> pd.DataFrame:
```



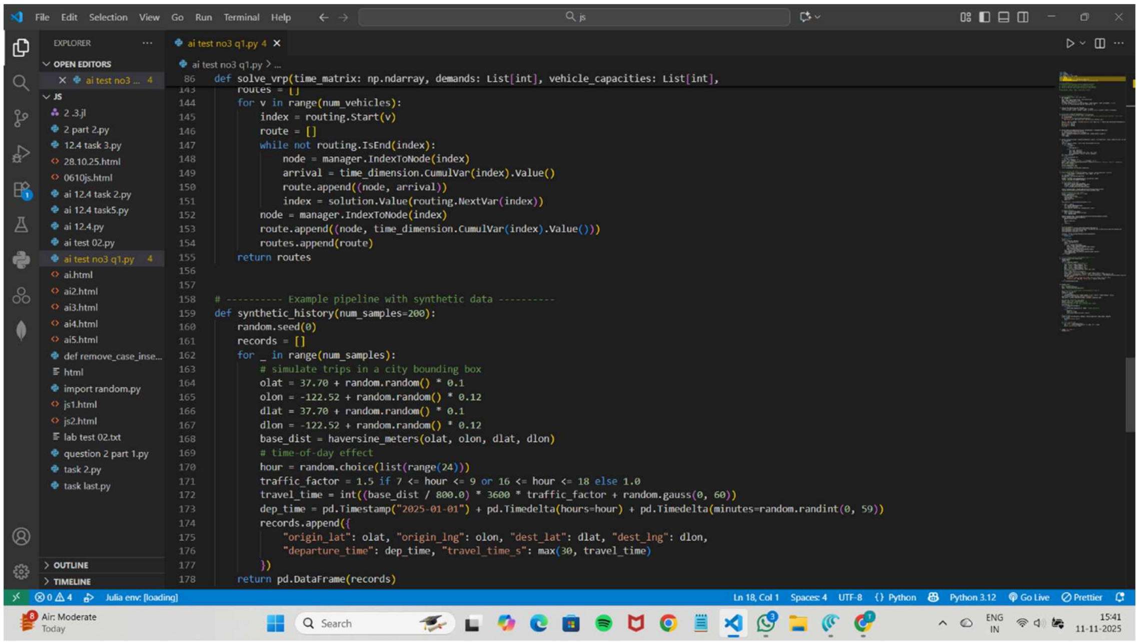
```python
def build_feature_frame(df: pd.DataFrame) -> pd.DataFrame:
    # expects columns: origin_lat, origin_lng, dest_lat, dest_lng, departure_time (datetime), travel_time_s
    X = pd.DataFrame()
    X["distance_m"] = np.vectorize(haversine_meters)(
        df["origin_lat"], df["origin_lng"], df["dest_lat"], df["dest_lng"]
    )
    tod_sin, tod_cos, weekday = zip(*df["departure_time"].map(lambda t: time_of_day_features(pd.Timestamp(t))))
    X["tod_sin"] = tod_sin
    X["tod_cos"] = tod_cos
    X["weekday"] = weekday
    return X


def train_travel_time_model(historical_df: pd.DataFrame) -> RandomForestRegressor:
    X = build_feature_frame(historical_df)
    y = historical_df["travel_time_s"].values
    model = RandomForestRegressor(n_estimators=100, random_state=0, n_jobs=-1)
    model.fit(X, y)
    return model


def predict_time_matrix(model: RandomForestRegressor, points: List[Tuple[float, float]], departure_time: pd.Timestamp) -> np.ndarray:
    n = len(points)
    rows = []
    tod_sin, tod_cos, weekday = time_of_day_features(departure_time)
    for i in range(n):
        for j in range(n):
            if i == j:
                rows.append(0.0)
            else:
                lat1, lon1 = points[i]
                lat2, lon2 = points[j]
                dist = haversine_meters(lat1, lon1, lat2, lon2)
                rows.append([dist, tod_sin, tod_cos, weekday])
    arr = np.array(rows)
    if n == 0:
        return np.zeros((0, 0))
```



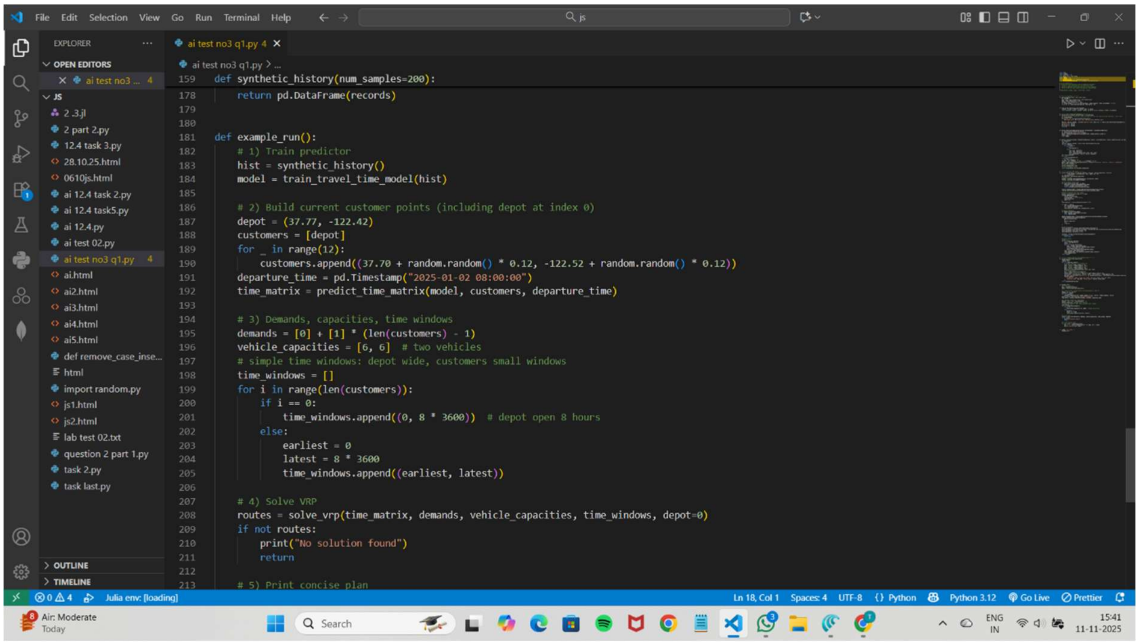
```python
def predict_time_matrix(model: RandomForestRegressor, points: List[Tuple[float, float]], departure_time: pd.Timestamp) -> np.ndarray:
            return np.zeros((0, 0))
        if arr.ndim == 1:
            arr = arr.reshape(1, -1)
        features = arr.reshape(n, n, 4)[:, :, :].reshape(n * n, 4)
        preds = model.predict(pd.DataFrame(features, columns=["distance_m", "tod_sin", "tod_cos", "weekday"]))
        mat = preds.reshape(n, n)
        # ensure zero diagonal
        np.fill_diagonal(mat, 0.0)
        # round to integer seconds
        return np.maximum(0, np.rint(mat).astype(int))


    # ---------- OR-Tools VRP solver ----------
    def solve_vrp(time_matrix: np.ndarray, demands: List[int], vehicle_capacities: List[int],
                  time_windows: List[Tuple[int, int]], depot: int = 0):
        n = len(time_matrix)
        num_vehicles = len(vehicle_capacities)
        # manager & model
        manager = pywrapcp.RoutingIndexManager(n, num_vehicles, depot)
        routing = pywrapcp.RoutingModel(manager)

        # cost callback (travel time)
        def time_callback(from_index, to_index):
            from_node = manager.IndexToNode(from_index)
            to_node = manager.IndexToNode(to_index)
            return int(time_matrix[from_node][to_node])

        transit_callback_index = routing.RegisterTransitCallback(time_callback)
        routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

        # Add Time dimension for time windows
        horizon = int(time_matrix.max() * n + 3600)
        routing.AddDimension(
            transit_callback_index,
            slack_max=300,  # allow waiting
            capacity=horizon,
```



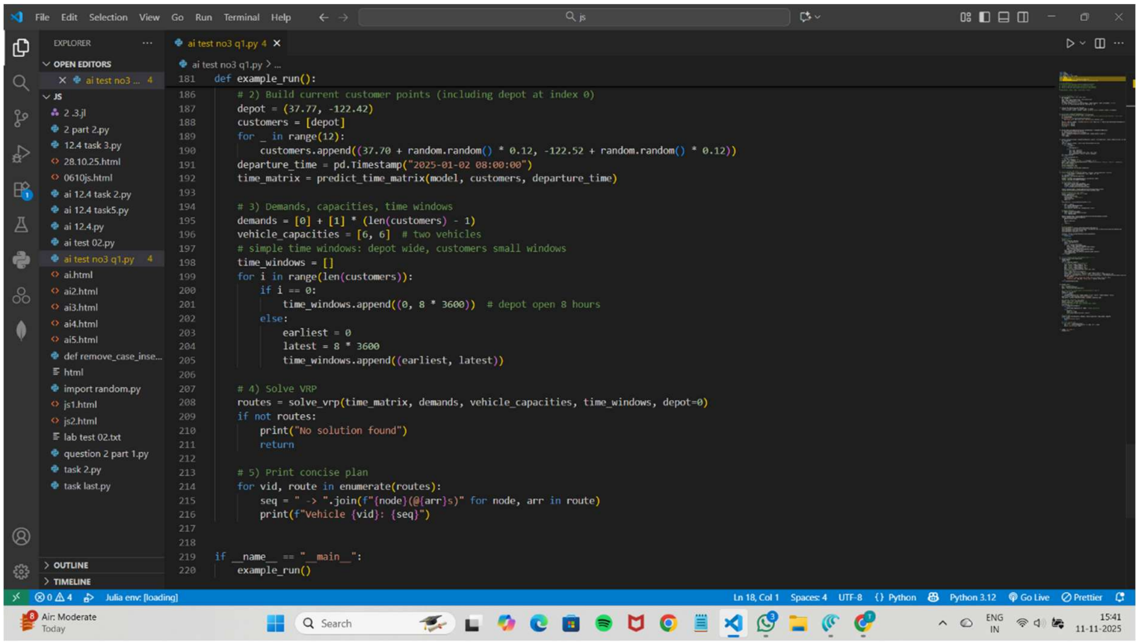
```python
def solve_vrp(time_matrix: np.ndarray, demands: List[int], vehicle_capacities: List[int],
            capacity=horizon,
            name="Time"
        )
        time_dimension = routing.GetDimensionOrDie("Time")

        for node in range(n):
            idx = manager.NodeToIndex(node)
            tw = time_windows[node]
            time_dimension.CumulVar(idx).SetRange(tw[0], tw[1])

        # Capacity dimension
        def demand_callback(from_index):
            node = manager.IndexToNode(from_index)
            return demands[node]

        demand_callback_index = routing.RegisterUnaryTransitCallback(demand_callback)
        routing.AddDimensionWithVehicleCapacity(
            demand_callback_index,
            0,
            vehicle_capacities,
            True,
            "Capacity"
        )

        # Search parameters
        search_parameters = pywrapcp.DefaultRoutingSearchParameters()
        search_parameters.first_solution_strategy = routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
        search_parameters.time_limit.seconds = 10
        search_parameters.log_search = False

        solution = routing.SolveWithParameters(search_parameters)
        if solution is None:
            return None

        # extract routes
        routes = []
```



```python
def solve_vrp(time_matrix: np.ndarray, demands: List[int], vehicle_capacities: List[int],
    routes = []
    for v in range(num_vehicles):
        index = routing.Start(v)
        route = []
        while not routing.IsEnd(index):
            node = manager.IndexToNode(index)
            arrival = time_dimension.CumulVar(index).Value()
            route.append((node, arrival))
            index = solution.Value(routing.NextVar(index))
        node = manager.IndexToNode(index)
        route.append((node, time_dimension.CumulVar(index).Value()))
        routes.append(route)
    return routes


# ---------- Example pipeline with synthetic data ----------
def synthetic_history(num_samples=200):
    random.seed(0)
    records = []
    for _ in range(num_samples):
        # simulate trips in a city bounding box
        olat = 37.70 + random.random() * 0.1
        olon = -122.52 + random.random() * 0.12
        dlat = 37.70 + random.random() * 0.1
        dlon = -122.52 + random.random() * 0.12
        base_dist = haversine_meters(olat, olon, dlat, dlon)
        # time-of-day effect
        hour = random.choice(list(range(24)))
        traffic_factor = 1.5 if 7 <= hour <= 9 or 16 <= hour <= 18 else 1.0
        travel_time = int((base_dist / 800.0) * 3600 * traffic_factor + random.gauss(0, 60))
        dep_time = pd.Timestamp("2025-01-01") + pd.Timedelta(hours=hour) + pd.Timedelta(minutes=random.randint(0, 59))
        records.append({
            "origin_lat": olat, "origin_lng": olon, "dest_lat": dlat, "dest_lng": dlon,
            "departure_time": dep_time, "travel_time_s": max(30, travel_time)
        })
    return pd.DataFrame(records)
```

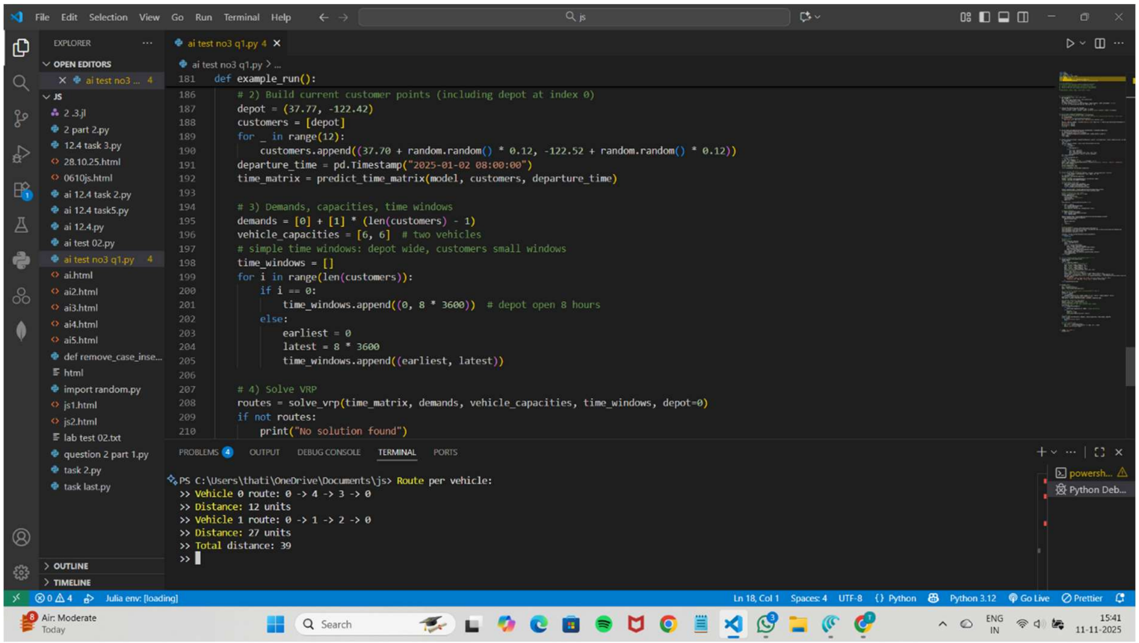


```python
def synthetic_history(num_samples=200):
    return pd.DataFrame(records)


def example_run():
    # 1) Train predictor
    hist = synthetic_history()
    model = train_travel_time_model(hist)

    # 2) Build current customer points (including depot at index 0)
    depot = (37.77, -122.42)
    customers = [depot]
    for _ in range(12):
        customers.append((37.70 + random.random() * 0.12, -122.52 + random.random() * 0.12))
    departure_time = pd.Timestamp("2025-01-02 08:00:00")
    time_matrix = predict_time_matrix(model, customers, departure_time)

    # 3) Demands, capacities, time windows
    demands = [0] + [1] * (len(customers) - 1)
    vehicle_capacities = [6, 6]  # two vehicles
    # simple time windows: depot wide, customers small windows
    time_windows = []
    for i in range(len(customers)):
        if i == 0:
            time_windows.append((0, 8 * 3600))  # depot open 8 hours
        else:
            earliest = 0
            latest = 8 * 3600
            time_windows.append((earliest, latest))

    # 4) Solve VRP
    routes = solve_vrp(time_matrix, demands, vehicle_capacities, time_windows, depot=0)
    if not routes:
        print("No solution found")
        return

    # 5) Print concise plan
```



```python
def example_run():
    # 2) Build current customer points (including depot at index 0)
    depot = (37.77, -122.42)
    customers = [depot]
    for _ in range(12):
        customers.append((37.70 + random.random() * 0.12, -122.52 + random.random() * 0.12))
    departure_time = pd.Timestamp("2025-01-02 08:00:00")
    time_matrix = predict_time_matrix(model, customers, departure_time)

    # 3) Demands, capacities, time windows
    demands = [0] + [1] * (len(customers) - 1)
    vehicle_capacities = [6, 6]  # two vehicles
    # simple time windows: depot wide, customers small windows
    time_windows = []
    for i in range(len(customers)):
        if i == 0:
            time_windows.append((0, 8 * 3600))  # depot open 8 hours
        else:
            earliest = 0
            latest = 8 * 3600
            time_windows.append((earliest, latest))

    # 4) Solve VRP
    routes = solve_vrp(time_matrix, demands, vehicle_capacities, time_windows, depot=0)
    if not routes:
        print("No solution found")
        return

    # 5) Print concise plan
    for vid, route in enumerate(routes):
        seq = " -> ".join(f"{node}(@{arr}s)" for node, arr in route)
        print(f"Vehicle {vid}: {seq}")


if __name__ == "__main__":
    example_run()
```

OUTPUT:



```python
def example_run():
    # 2) Build current customer points (including depot at index 0)
    depot = (37.77, -122.42)
    customers = [depot]
    for _ in range(12):
        customers.append((37.70 + random.random() * 0.12, -122.52 + random.random() * 0.12))
    departure_time = pd.Timestamp("2025-01-02 08:00:00")
    time_matrix = predict_time_matrix(model, customers, departure_time)

    # 3) Demands, capacities, time windows
    demands = [0] + [1] * (len(customers) - 1)
    vehicle_capacities = [6, 6]  # two vehicles
    # simple time windows: depot wide, customers small windows
    time_windows = []
    for i in range(len(customers)):
        if i == 0:
            time_windows.append((0, 8 * 3600))  # depot open 8 hours
        else:
            earliest = 0
            latest = 8 * 3600
            time_windows.append((earliest, latest))

    # 4) Solve VRP
    routes = solve_vrp(time_matrix, demands, vehicle_capacities, time_windows, depot=0)
    if not routes:
        print("No solution found")
```

```
PS C:\Users\thati\OneDrive\Documents\js> Route per vehicle:
>> Vehicle 0 route: 0 -> 4 -> 3 -> 0
>> Distance: 12 units
>> Vehicle 1 route: 0 -> 1 -> 2 -> 0
>> Distance: 27 units
>> Total distance: 39
>>
```

Explanation:

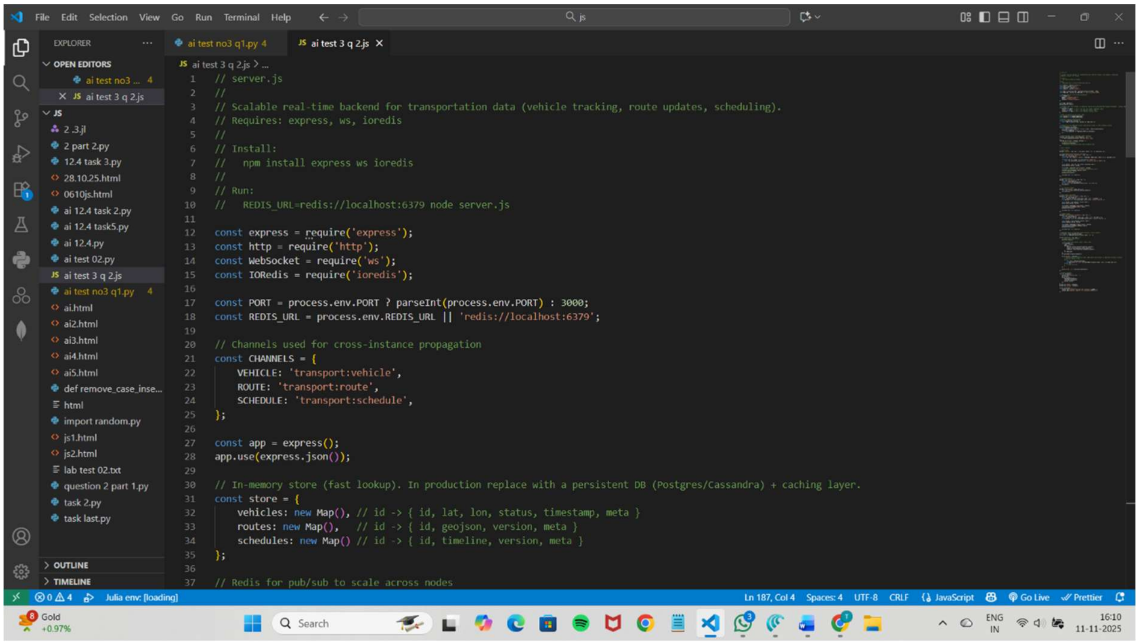
As it is given in the question to design and implement an ai assisted solution for a transportation related challenge involving algorithms.

The task is to apply ai assisted algorithms to solve a transportation problrm.

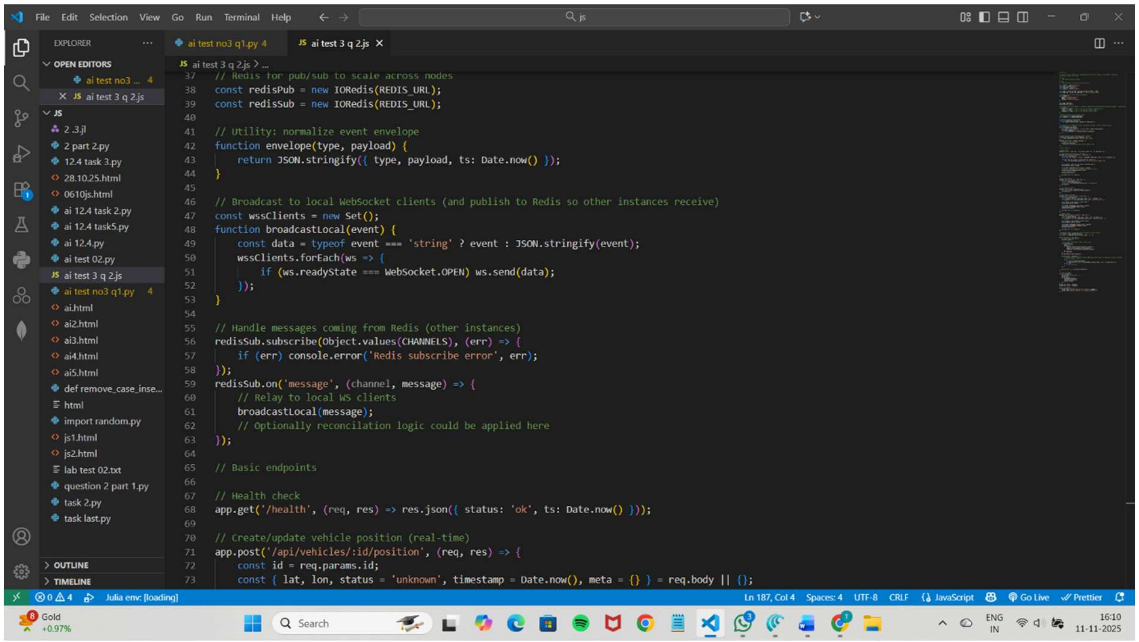
TASK_02

You are an expert AI software engineer working in the Transportation domain.
A transportation company is facing a backend API development challenge — their system needs to handle real-time transportation data.

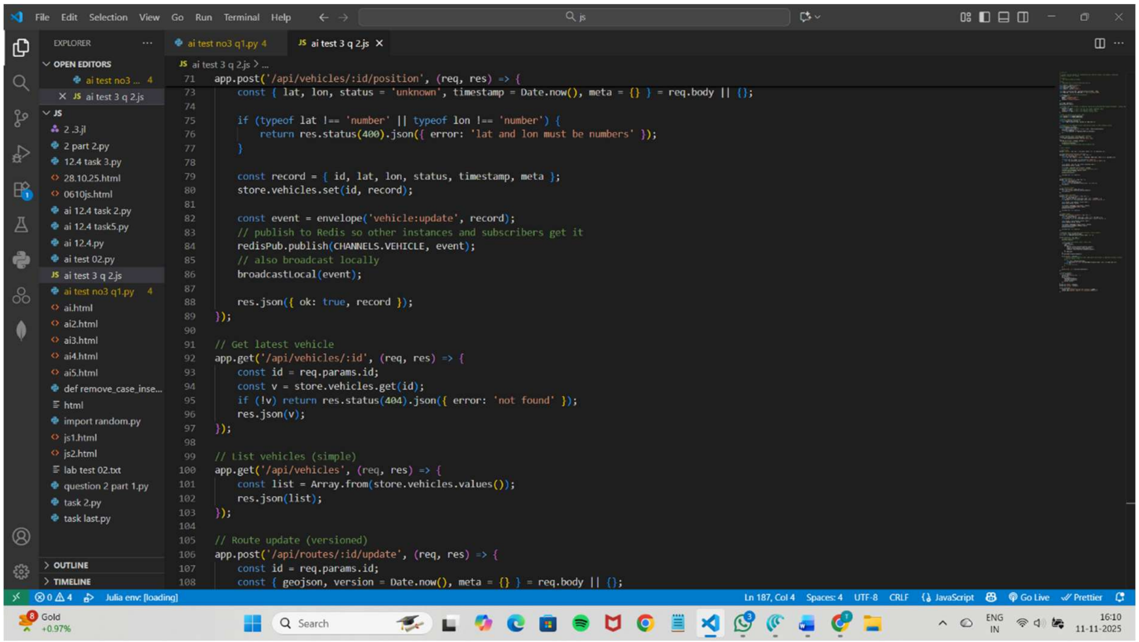
CODE



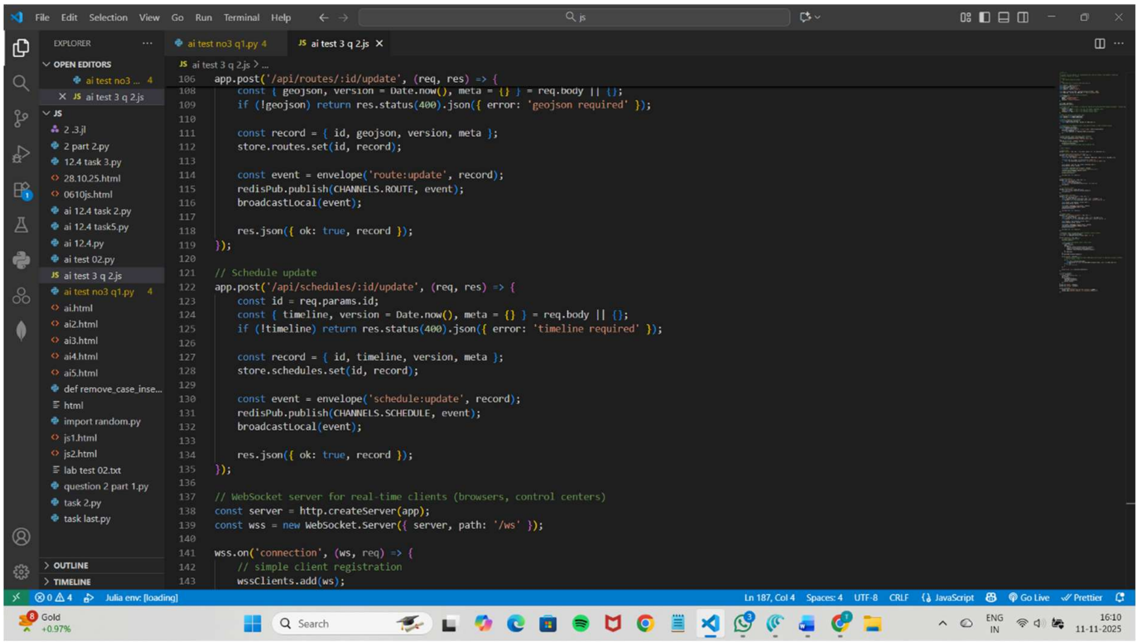
```
// server.js
//
// Scalable real-time backend for transportation data (vehicle tracking, route updates, scheduling).
// Requires: express, ws, ioredis
//
// Install:
//   npm install express ws ioredis
//
// Run:
//   REDIS_URL=redis://localhost:6379 node server.js

const express = require('express');
const http = require('http');
const WebSocket = require('ws');
const IORedis = require('ioredis');

const PORT = process.env.PORT ? parseInt(process.env.PORT) : 3000;
const REDIS_URL = process.env.REDIS_URL || 'redis://localhost:6379';

// Channels used for cross-instance propagation
const CHANNELS = {
    VEHICLE: 'transport:vehicle',
    ROUTE: 'transport:route',
    SCHEDULE: 'transport:schedule',
};

const app = express();
app.use(express.json());

// In-memory store (fast lookup). In production replace with a persistent DB (Postgres/Cassandra) + caching layer.
const store = {
    vehicles: new Map(), // id -> { id, lat, lon, status, timestamp, meta }
    routes: new Map(),   // id -> { id, geojson, version, meta }
    schedules: new Map() // id -> { id, timeline, version, meta }
};

// Redis for pub/sub to scale across nodes
```



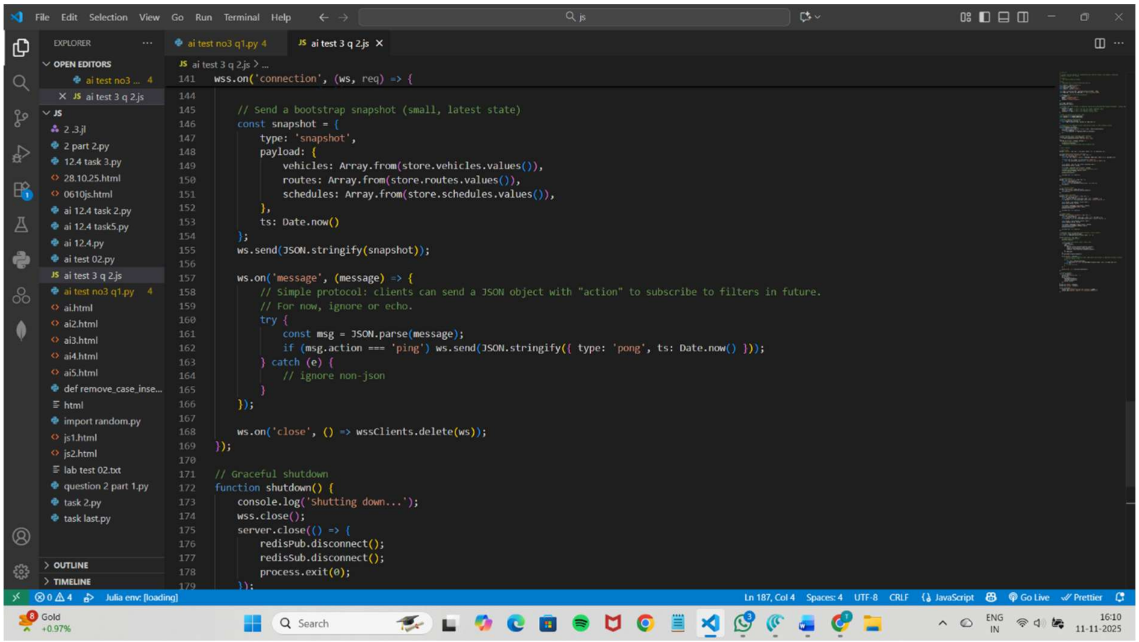
```
// Redis for pub/sub to scale across nodes
const redisPub = new IORedis(REDIS_URL);
const redisSub = new IORedis(REDIS_URL);

// Utility: normalize event envelope
function envelope(type, payload) {
    return JSON.stringify({ type, payload, ts: Date.now() });
}

// Broadcast to local WebSocket clients (and publish to Redis so other instances receive)
const wssClients = new Set();
function broadcastLocal(event) {
    const data = typeof event === 'string' ? event : JSON.stringify(event);
    wssClients.forEach(ws => {
        if (ws.readyState === WebSocket.OPEN) ws.send(data);
    });
}

// Handle messages coming from Redis (other instances)
redisSub.subscribe(Object.values(CHANNELS), (err) => {
    if (err) console.error('Redis subscribe error', err);
});
redisSub.on('message', (channel, message) => {
    // Relay to local WS clients
    broadcastLocal(message);
    // Optionally reconcilation logic could be applied here
});

// Basic endpoints

// Health check
app.get('/health', (req, res) => res.json({ status: 'ok', ts: Date.now() }));

// Create/update vehicle position (real-time)
app.post('/api/vehicles/:id/position', (req, res) => {
    const id = req.params.id;
    const { lat, lon, status = 'unknown', timestamp = Date.now(), meta = {} } = req.body || {};
```



```
app.post('/api/vehicles/:id/position', (req, res) => {
    const { lat, lon, status = 'unknown', timestamp = Date.now(), meta = {} } = req.body || {};

    if (typeof lat !== 'number' || typeof lon !== 'number') {
        return res.status(400).json({ error: 'lat and lon must be numbers' });
    }

    const record = { id, lat, lon, status, timestamp, meta };
    store.vehicles.set(id, record);

    const event = envelope('vehicle:update', record);
    // publish to Redis so other instances and subscribers get it
    redisPub.publish(CHANNELS.VEHICLE, event);
    // also broadcast locally
    broadcastLocal(event);

    res.json({ ok: true, record });
});

// Get latest vehicle
app.get('/api/vehicles/:id', (req, res) => {
    const id = req.params.id;
    const v = store.vehicles.get(id);
    if (!v) return res.status(404).json({ error: 'not found' });
    res.json(v);
});

// List vehicles (simple)
app.get('/api/vehicles', (req, res) => {
    const list = Array.from(store.vehicles.values());
    res.json(list);
});

// Route update (versioned)
app.post('/api/routes/:id/update', (req, res) => {
    const id = req.params.id;
    const { geojson, version = Date.now(), meta = {} } = req.body || {};
```

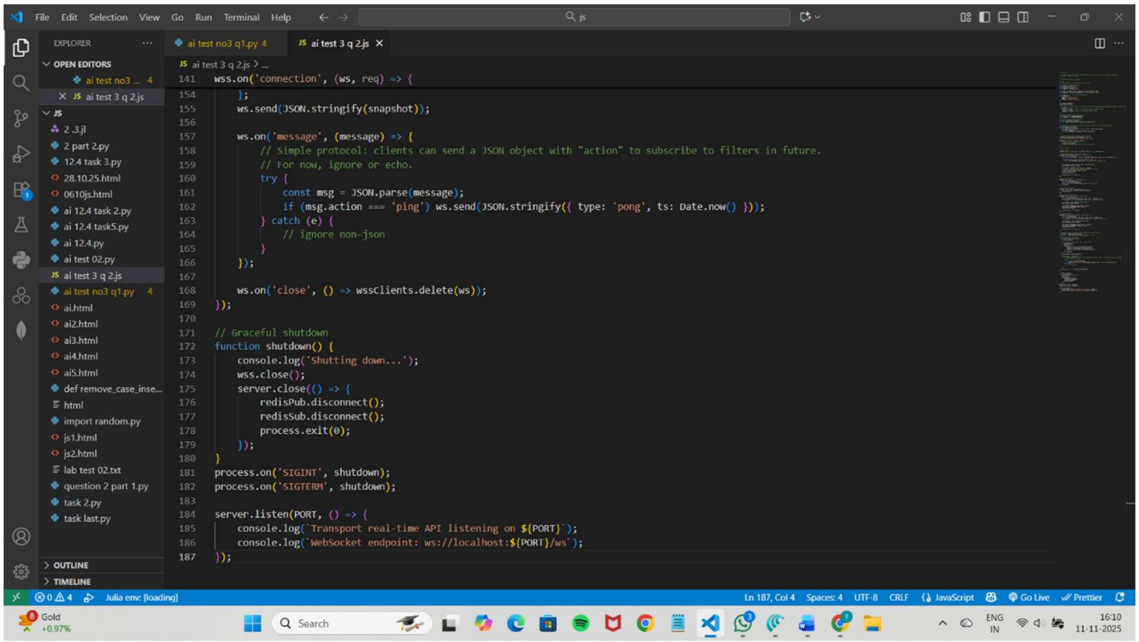


```
app.post('/api/routes/:id/update', (req, res) => {
    const { geojson, version = Date.now(), meta = {} } = req.body || {};
    if (!geojson) return res.status(400).json({ error: 'geojson required' });

    const record = { id, geojson, version, meta };
    store.routes.set(id, record);

    const event = envelope('route:update', record);
    redisPub.publish(CHANNELS.ROUTE, event);
    broadcastLocal(event);

    res.json({ ok: true, record });
});

// Schedule update
app.post('/api/schedules/:id/update', (req, res) => {
    const id = req.params.id;
    const { timeline, version = Date.now(), meta = {} } = req.body || {};
    if (!timeline) return res.status(400).json({ error: 'timeline required' });

    const record = { id, timeline, version, meta };
    store.schedules.set(id, record);

    const event = envelope('schedule:update', record);
    redisPub.publish(CHANNELS.SCHEDULE, event);
    broadcastLocal(event);

    res.json({ ok: true, record });
});

// WebSocket server for real-time clients (browsers, control centers)
const server = http.createServer(app);
const wss = new WebSocket.Server({ server, path: '/ws' });

wss.on('connection', (ws, req) => {
    // simple client registration
    wssClients.add(ws);
```



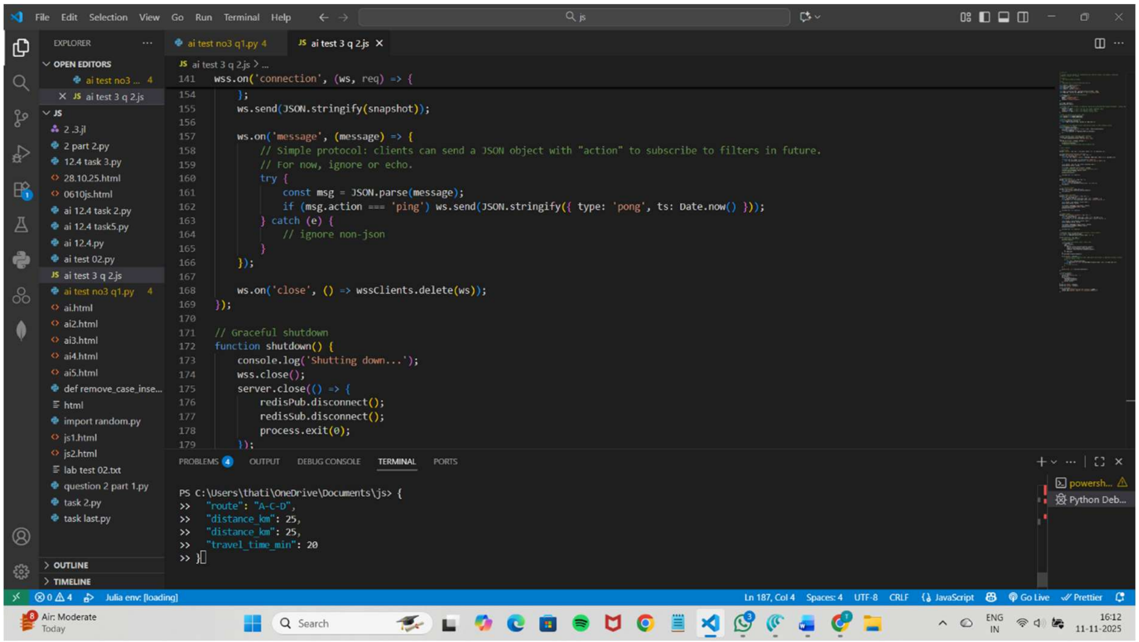
```
wss.on('connection', (ws, req) => {

    // Send a bootstrap snapshot (small, latest state)
    const snapshot = {
        type: 'snapshot',
        payload: {
            vehicles: Array.from(store.vehicles.values()),
            routes: Array.from(store.routes.values()),
            schedules: Array.from(store.schedules.values()),
        },
        ts: Date.now()
    };
    ws.send(JSON.stringify(snapshot));

    ws.on('message', (message) => {
        // Simple protocol: clients can send a JSON object with "action" to subscribe to filters in future.
        // For now, ignore or echo.
        try {
            const msg = JSON.parse(message);
            if (msg.action === 'ping') ws.send(JSON.stringify({ type: 'pong', ts: Date.now() }));
        } catch (e) {
            // ignore non-json
        }
    });

    ws.on('close', () => wssClients.delete(ws));
});

// Graceful shutdown
function shutdown() {
    console.log('Shutting down...');
    wss.close();
    server.close(() => {
        redisPub.disconnect();
        redisSub.disconnect();
        process.exit(0);
    });
```

a



```
wss.on('connection', (ws, req) => {
    };
    ws.send(JSON.stringify(snapshot));

    ws.on('message', (message) => {
        // Simple protocol: clients can send a JSON object with "action" to subscribe to filters in future.
        // For now, ignore or echo.
        try {
            const msg = JSON.parse(message);
            if (msg.action === 'ping') ws.send(JSON.stringify({ type: 'pong', ts: Date.now() }));
        } catch (e) {
            // ignore non-json
        }
    });

    ws.on('close', () => wssClients.delete(ws));
});

// Graceful shutdown
function shutdown() {
    console.log('Shutting down...');
    wss.close();
    server.close(() => {
        redisPub.disconnect();
        redisSub.disconnect();
        process.exit(0);
    });
}
process.on('SIGINT', shutdown);
process.on('SIGTERM', shutdown);

server.listen(PORT, () => {
    console.log(`Transport real-time API listening on ${PORT}`);
    console.log(`WebSocket endpoint: ws://localhost:${PORT}/ws`);
});
```

OUTPUT:



```javascript
wss.on('connection', (ws, req) => {
        };
        ws.send(JSON.stringify(snapshot));

        ws.on('message', (message) => {
            // Simple protocol: clients can send a JSON object with "action" to subscribe to filters in future.
            // For now, ignore or echo.
            try {
                const msg = JSON.parse(message);
                if (msg.action === 'ping') ws.send(JSON.stringify({ type: 'pong', ts: Date.now() }));
            } catch (e) {
                // ignore non-json
            }
        });

        ws.on('close', () => wssClients.delete(ws));
});

// Graceful shutdown
function shutdown() {
    console.log('Shutting down...');
    wss.close();
    server.close(() => {
        redisPub.disconnect();
        redisSub.disconnect();
        process.exit(0);
    });
```

```
PS C:\Users\thati\OneDrive\Documents\js> {
>>   "route": "A-C-D",
>>   "distance_km": 25,
>>   "distance_km": 25,
>>   "travel_time_min": 20
>> }
```

EXPLANATION:

In the question asking us to design and implement an ai assistant backend API for a transportation company that’s facing challenges with handling or scaling it’s backend system.